Real-Parameter Black-Box Optimisation: Benchmarking and Designing Algorithms

Raymond Ros

N° d'ordre : 9743

Thèse

de L'Université Paris–Sud

presentée en vue de l'obtention du grade de

Docteur de l'Université Paris–Sud

Specialité : Informatique

par

Raymond ROS

Optimisation Continue Boîte Noire : Comparaison et Conception d'Algorithmes

Soutenue le 21 Décembre 2009 devant la commission d'examen :

M.	Frédéric BONNANS	INRIA	Examinateur
M.	Nikolaus HANSEN	INRIA	Co-Directeur de thèse
M.	Rodolphe LE RICHE	CNRS	Rapporteur
M.	Günter RUDOLPH	TU Dortmund	Rapporteur
Mme	Michèle SEBAG	CNRS	Directrice de thèse

Laboratoire de Recherche en Informatique, U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France

Résumé

Introduction

Un problème d'optimisation peut se formaliser en la recherche d'un élément $\boldsymbol{x}_{\text{opt}}$ du domaine Ω sous-ensemble de $\mathbb{R}^n$ qui minimisera la *fonction objectif* $f : \Omega \to \mathbb{R}$. En optimisation continue, où les composants des vecteurs solutions $\boldsymbol{x}$ sont à valeurs réelles, le scénario *boîte noire* fait l'hypothèse que l'évaluation de la fonction objectif est l'unique moyen de collecter des informations sur le problème considéré. Nous nous intéressons à la question du choix d'un algorithme pour résoudre de tels problèmes d'optimisation boîte noire.

De nombreux algorithmes existent dans les domaines de l'optimisation globale, de la recherche opérationnelle ou encore de l'évolution artificielle. Comment pouvons-nous comparer ces algorithmes de manière à en choisir un qui soit approprié pour résoudre un problème d'optimisation donné ? Cette question sera traitée dans ce manuscrit.

La théorie ne propose pas de réponse à cette question dans la mesure où nous faisons l'hypothèse de ne recéler aucune information sur la fonction objectif. Les résultats théoriques existants utilisent des hypothèses simplificatrices qui ne sont plus valides face à des problèmes réels.

Des procédures systématiques pour la comparaison d'algorithmes existent et nous fournissent des éléments de réponse. Nous regroupons ces procédures systématiques sous l'appellation *benchmarking*. Nos comparaisons systématiques reposent sur la notion de *coûts de l'optimisation* que nous définissons comme la quantité de calculs nécessaires pour qu'un algorithme atteigne la solution.

Un élément critique dans la comparaison systématique d'algorithmes est l'ensemble des problèmes qui serviront à la comparaison. Dans la mesure où il n'est pas

possible de traiter l'ensemble des problèmes du monde réel, nous tâchons de comparer les algorithmes sur un ensemble de fonctions artificielles qui possèdent des propriétés reconnues difficiles en optimisation. Ainsi nous pourrons déterminer les points forts et faibles d'algorithmes vis-à-vis de ces propriétés.

L'étude des algorithmes dans ce contexte nous permettra :

- dans le cas où des propriétés du problème d'optimisation considéré sont connues, de proposer des algorithmes dont les points forts y correspondent,
- à l'inverse si aucune information n'est disponible, l'analyse des performances de quelques algorithmes choisis à la lumière des résultats du benchmarking peut permettre d'obtenir des renseignements sur le problème.

Le fléau de la dimension est une propriété correspondant à l'explosion exponentielle du volume de l'espace de recherche tandis que sa dimension n grandit : des techniques d'optimisation appropriées pour n petit peuvent ne pas convenir pour n modéré ou grand. Ceci est une motivation forte pour étudier l'extensibilité des algorithmes, c'est à dire leurs performances par rapport à n.

Le second chapitre de ce manuscrit fait l'état de l'art des méthodes pour l'optimisation boîte noire ainsi que dans le domaine du benchmarking en optimisation continue. En l'occurrence, il est montré qu'il existe des articles, des outils de comparaison systématique dans les domaines de l'optimisation globale, de la recherche opérationnelle et de l'évolution artificielle. Cependant, ces comparaisons n'impliquent qu'un nombre peu important d'algorithmes ou de problèmes. Aussi il n'existe que peu de solutions logicielles pour le benchmarking.

Disposer d'un logiciel pour le benchmarking d'algorithmes permettrait d'obtenir plus facilement des résultats d'algorithmes sur un nombre conséquent de fonctions. A la lumière de ces résultats de comparaison, l'utilisateur pourra décider quel algorithme choisir pour optimiser une fonction dont il connait les propriétés. Inversement, l'utilisateur face à une fonction inconnue pourra obtenir des informations en comparant les résultats d'algorithmes sur cette fonction à ceux du benchmarking.

Ce manuscrit présente nos contributions. Le chapitre 3 se focalise sur l'algorithme Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) et décrit des variantes appropriées aux problèmes à grande dimensionalité. Les chapitres 4 et 5 présentent nos résultats de comparaisons d'algorithmes et notre implémentation logicielle COCO.

Notions fondamentales

Optimisation continue et benchmarking

Nous choisissons d'étudier des fonctions artificielles. Les fonctions que nous choisissons mettent en évidence certaines propriétés qui se retrouvent dans les problèmes réels difficiles :

la grande dimensionalité, l'explosion exponentielle du volume de l'espace de recherche avec la dimension du problème rend la recherche aléatoire intraitable,

la non-séparabilité, les dépendances entre paramètres sont essentielles pour l'optimisation de la fonction objectif, il est possible de rendre non-séparable une fonction séparable en effectuant simplement une rotation de l'espace de recherche,

la multi-modalité, la présence d'optima locaux peut gêner la résolution du problème par des techniques d'optimisation locales,

la rugosité, la fonction objectif représente un paysage très irrégulier,

la non-convexité qui peut affecter les algorithmes à région de confiance par exemple puisqu'ils interpolent la fonction par un modèle convexe,

le mauvais conditionnement, le ratio entre une variation de la fonction et une variation de l'argument est grand, ce qui peut entraîner des problèmes numériques,

le bruit ajoute de l'incertitude à l'évaluation de la fonction et peut se traduire par de la rugosité.

La notion de performance des algorithmes testés sera liée au nombre d'évaluations de la fonction objectif. Etant donné une valeur cible de la fonction objectif, f_{target}, nous pouvons définir un temps d'exécution pour atteindre cette valeur cible comme étant le nombre d'évaluations de la fonction objectif pour obtenir une valeur plus petite que f_{target}. Si nous testons un algorithme plusieurs fois sur la même fonction, nous pouvons définir :

$\mathbf{RT_S}$ le temps d'exécution moyen pour les tentatives ayant atteint f_{target} avec succès,

$\mathbf{RT_{US}}$ le temps d'exécution moyen pour les tentatives n'ayant pas atteint f_{target}, nous considérons dans ce cas là le temps d'exécution final,

$p_{\mathbf{S}}$ le taux de succès,

$SP1 = \frac{\mathbf{RT_S}}{p_{\mathbf{S}}}$ estime le temps d'exécution pour qu'un algorithme relancé à chaque échec atteigne une fois f_{target}, en faisant l'hypothèse que le temps d'exécution moyen des tentatives qui échouent est égal à celui des tentatives qui réussissent,

$\text{ERT} = \frac{p_{\mathbf{S}}\mathbf{RT_S}+(1-p_{\mathbf{S}})\mathbf{RT_{US}}}{p_{\mathbf{S}}}$ estime le temps d'exécution pour qu'un algorithme relancé à chaque échec atteigne une fois f_{target}, aucune hypothèse n'étant faite sur le temps d'exécution moyen des tentatives qui échouent.

Algorithmes d'optimisation continue

Nous présentons une liste de différentes méthodes des domaines de la recherche opérationnelle, l'optimisation globale, l'évolution artificielle. Parmi ces méthodes, on peut trouver des approches théoriques, pour lesquelles la convergence est prouvée sous certaines conditions, ainsi que des heuristiques empiriques qui s'intéressent davantage à trouver des solutions en pratique.

Covariance Matrix Adaptation-Evolution Strategy (CMA-ES)

L'algorithme CMA-ES [Hansen and Kern, 2004, Hansen and Ostermeier, 2001, Hansen et al., 2003] utilise un mécanisme d'adaptation de la matrice de covariance couplé à une stratégie d'évolution.

La recherche de l'optimum dans CMA-ES passe par l'équation suivante :

$$\boldsymbol{x}_k^{(g+1)} \sim \langle \boldsymbol{x} \rangle_{\text{W}}^{(g)} + \sigma^{(g)} \underbrace{\boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)}\boldsymbol{z}_k^{(g+1)}}_{\mathcal{N}\left(0,\boldsymbol{C}^{(g)}\right)}, \; k = 1, \ldots, \lambda$$

avec λ étant la taille de la population, $\boldsymbol{x}_k^{(g+1)}$ les individus générés à la génération $g+1$. La distribution de ces individus est donnée par une loi de distribution normale multidimensionnelle avec les paramètres $\langle \boldsymbol{x} \rangle_{\text{W}}^{(g)}$, $\sigma^{(g)}$ et $\boldsymbol{C}^{(g)}$.

La matrice de covariance $\boldsymbol{C}^{(g)}$ est mise à jour suivant l'équation :

$$\boldsymbol{C}^{(g+1)} = (1 - c_{\text{cov}})\boldsymbol{C}^{(g)} + \frac{1}{\mu_{\text{cov}}} c_{\text{cov}} \boldsymbol{p}_{\text{c}}{}^{(g+1)} \left(\boldsymbol{p}_{\text{c}}{}^{(g+1)}\right)^T + c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \underbrace{\sum_{i=1}^{\mu} w_i \boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} \boldsymbol{z}_{i:\lambda}^{(g+1)} \left(\boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} \boldsymbol{z}_{i:\lambda}^{(g+1)}\right)^T}_{\sum_{i=1}^{\mu} \frac{w_i}{\sigma^{(g)2}} \left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \langle \boldsymbol{x} \rangle_{\text{W}}^{(g)}\right) \left(\boldsymbol{x}_{i:\lambda}^{(g+1)} - \langle \boldsymbol{x} \rangle_{\text{W}}^{(g)}\right)^T}$$

Cette équation dépend de trois termes, l'équilibre entre le premier et les deux autres termes est contrôlé par le taux d'apprentissage $c_{\text{cov}} \in [0, 1]$. Un taux d'apprentissage nul fixerait la matrice de covariance à sa valeur initiale.

Autres algorithmes

Nous mentionnons dans ce manuscrit de nombreux algorithmes (Chap. 2). En particulier nous nous sommes intéressés aux algorithmes d'Evolution Différentielle (DE) et d'optimisation par essaims de particules (PSO). Ces deux algorithmes évolutionnaires utilisent des populations qui sont mises à jour sur des principes d'inerties en fonction des meilleurs individus et de pertubations appliquées à ces individus.

L'algorithme classique BFGS est une méthode à directions de descente de quasi-Newton. NEWUOA est un algorithme à régions de confiance.

Variantes de CMA-ES

La dimension de l'espace de recherche, n, joue un rôle essentiel dans l'optimisation continue à cause du *fléau de la dimension*. Ceci n'est vrai que dans le cas de fonctions où les variables montrent des dépendances entre elles. Dans le domaine de l'évolution artificielle, l'adaptation de la matrice de covariance permet d'apprendre ces dépendances avec succès. L'algorithme Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) apprend les dépendances entre paramètres de la fonction et a une complexité algorithmique d'au moins $\mathcal{O}(n^2)$. Des résultats empiriques montrent que pour apprendre la matrice de covariance complète, l'algorithme a besoin d'un nombre d'évaluations de la fonction objectif (ou aussi coût de l'optimisation) en général au plus quadratique par rapport à n [Hansen and Ostermeier, 2001, Hansen et al., 2003].

Le nombre de paramètres internes $\frac{n^2+n}{2}$ de CMA-ES ainsi que sa complexité algorithmique constituent des limitations dans le cas de fonctions où n est grand.

Nous proposons des variantes de CMA-ES qui font diminuer ce nombre de paramètres pour ainsi privilégier un coût moindre de l'apprentissage au détriment de la complexité du modèle. Dans ces variantes, sep-CMA et block-CMA, permettent de n'apprendre que les blocs diagonaux de la matrice de covariance. Ceci est réalisé en fixant les autres termes à la valeur zéro lors de la mise à jour de la matrice de covariance par l'équation 3.5. Cette contrainte sur la matrice de covariance permet de modifier le taux d'apprentissage c_{cov} qui est inversement proportionnel au carré des degrés de liberté de la matrice de covariance.

La contrainte sur la matrice de covariance dans sep-CMA et block-CMA empêche d'apprendre l'ensemble des dépendances entre paramètres, mais favorise ces variantes sur les fonctions séparables.

La variante sep-CMA-ES a ainsi une complexité linéaire. Notre étude empirique sur l'effet de varier le taux d'apprentissage nous a permis de déterminer qu'il pouvait être multiplié par $\frac{n+3/2}{3}$. Une conséquence est une adaptation plus rapide des paramètres de la matrice de covariance. La variante block-CMA-ES permet de contrôler sa complexité en fonction de la configuration des blocs diagonaux. Le taux d'apprentissage peut aussi être augmenté.

Nous avons testé sep-CMA-ES et block-CMA-ES en terme de coût de l'optimisation ainsi qu'en terme de temps de calcul. Nous avons ainsi mis en évidence que sep-CMA-ES montrent des performances proportionnelles à la dimension sur des fonctions séparables. Les différentes configurations de block-CMA-ES permettent de passer graduellement des performances de sep-CMA-ES à celles de CMA-ES.

Les performances sur la fonction de Rosenbrock qui n'est pas séparable montrent que CMA-ES est plus rapide que sep-CMA-ES et block-CMA-ES. Par contre cela n'est plus le cas quand la dimension du problème est plus grande que 128. Ce résultat suggère que la fonction de Rosenbrock est *partiellement séparable*. Cette hypothèse est confirmée par le fait que sep-CMA-ES et block-CMA-ES sont plus lents que CMA-ES pour la fonction de Rosenbrock avec rotation.

Ces résultats valident l'intérêt d'utiliser sep-CMA-ES et block-CMA-ES pour des problèmes à grande dimensionalité.

Optimisation continue boîte noire

Nous proposons de comparer différents algorithmes de manière systématique sur des fonctions tests. Les résultats de ces comparaisons seront utiles dans au moins deux cas de figure. Les résultats permettrait de déterminer quel(s) algorithme(s) utiliser pour résoudre un problème partageant des similarités avec les fonctions tests. Par ailleurs, la comparaison des algorithmes peut mettre en lumière leurs points forts et faibles par rapport aux propriétés de fonctions.

Nous avons ainsi comparé des algorithmes des domaines de l'évolution artificielle, la recherche opérationnelle et des mathématiques déterministes. Les performances des algorithmes sont déterminées en fonction du nombre d'évaluations de la fonction test.

Dans les protocoles expérimentaux que nous utilisons, nous utilisons des stratégies de redémarrage d'algorithme. En théorie, avec un horizon infini, le redémarrage indépendant d'un algorithme stochastique permet d'avoir une probabilité de succès de 1. En pratique, le redémarrage dans la limite d'un budget initial permet d'augmenter le taux de succès de l'algorithme.

Nous présentons les résultats de deux instances de benchmarking.

Etude de la non-séparabilité, non-convexité et mauvais conditionnement

Nous avons testé les algorithmes CMA-ES, Evolution Différentielle (DE), Optimisation par essaim de particle (PSO), NEWUOA et BFGS.

Au travers de quelques fonctions test simples, nous avons pu étudié les propriétés :

non-séparabilité nous avons considéré des fonctions, pour certaines séparables, ainsi que ces mêmes fonctions avec une rotation de l'espace de recherche,

non-convexité nous avons étudié la fonction ellipsoïde qui est convexe quadratique ainsi que la fonction ellipsoïde à la puissance un quart qui n'est pas convexe,

mauvais conditionnement nous avons étudié la fonction ellipsoïde et Rosenbrock avec plusieurs valeurs de conditionnement allant de 1 à 10^10.

Il s'avère que :

- PSO et DE sont peu affectés par le conditionnement de la fonction ellipsoïde,

- CMA-ES et DE sont invariants par rotation de l'espace de recherche,
- CMA-ES, DE et PSO sont invariants par transformation préservant l'ordre,
- PSO est très affecté par la rotation de l'espace de recherche,
- NEWUOA et BFGS sont affectés par la non-convexité d'une fonction.

En particulier, nous observons que les algorithmes BFGS et NEWUOA n'ont des performances que meilleures d'un facteur de 5 par rapport à celles de CMA-ES sur des fonctions convexes quadratiques tournées qui constituent normalement les problèmes les plus faciles pour ces méthodes.

BBOB 2009

Nous avons organisé l'atelier de recherche Black-Box Optimisation Benchmarking (BBOB) pour la conférence internationale GECCO 2009. Nous y proposions un cadre expérimental et deux ensembles de fonctions bruitées et non bruitées, le tout étant accessible sous la forme d'un logiciel, COCO. Les chercheurs en optimisation continue purent soumettre les résultats d'algorithmes qu'ils avaient testés.

Nous compilons dans ce manuscrit les résultats de 32 algorithmes répartis en 45 articles scientifiques publiés dans les actes de la conférence. Nous avons en particulier testé les algorithmes NEWUOA, BFGS, et IPOP-sep-CMA-ES. Les deux premiers algorithmes utilisent des stratégies de redémarrages indépendants, tandis que le dernier utilise une stratégie avec accroissement de la taille de la population (IPOP).

Le protocole expérimental pour tester un algorithme consiste en 15 répétitions sur chacune des fonctions tests dans des dimensions allant de 2 à 40. Les fonctions tests proposées démontrent toutes des propriétés parmi celles que nous mentionnons plus haut. Les performances sont quantifiées en terme de nombre d'évaluations de fonction et en terme de temps de calcul par évaluation.

Sur l'ensemble des fonctions non bruitées nous montrons que le meilleur algorithme dépend du budget alloué : si le budget est inférieur quelques centaines d'évaluations fois la dimension NEWUOA démontre les meilleures performances tandis que IPOP-sep-CMA-ES est le meilleur devant NEWUOA et BFGS quand le budget est plus large.

Nous compilons dans ce manuscrit les résultats de beaucoup d'autres algoritmes soumis à BBOB. Nous observons que sur les fonctions non bruitées en dimension 2 et 3, l'algorithme du simplexe de Nelder-Mead est le meilleur. Sur les fonctions bruitées comme non bruitées les méthodes utilisant une adaptation de la matrice de covariance obtiennent les meilleurs résultats, la meilleure méthode étant la variante BIPOP-CMA-ES qui utilise en parallèle une petite et une grande population.

Ces deux études différentes ont permis de mettre en lumière les points forts et faibles d'algorithmes de différents champs de recherche, et en particulier donnent des résultats d'*extensibilité* des approches. Pour quantifier les performances des algorithmes dans notre protocole expérimental, il était nécessaire d'obtenir au moins un succès. Cela nous a permis de montrer l'intérêt des stratégies de redémarrage dans les algorithmes.

Logiciel de comparaison systématique : COCO

Pour l'atelier BBOB, une de nos contributions significatives a été de proposer un logiciel permettant de tester systématiquement des implémentations logicielles d'un algorithme d'optimisation continue sur un ensemble de fonctions tests.

Ce chapitre présente le fonctionnement de ce logiciel qui fournit :

- une interface logicielle fgeneric pour accéder aux fonctions tests, proposée dans plusieurs langages de programmation,
- un protocole expérimental implémenté,
- un module de post-traitement bbob_pproc générant figures et tableaux,
- des modèles de documents LaTeX présentant l'ensemble de ces résultats.

L'essentiel de la tâche pour l'utilisateur de COCO sera d'interfacer l'algorithmes qu'il veut tester avec l'interface fgeneric.

Des exemples de scripts exampleexperiment et exampletiming permettent de réaliser une expérience complète sur l'ensemble des fonctions tests. L'exécution du script exampleexperiment permet d'obtenir des données expérimentales qui seront traitées à l'aide dans bbob_pproc. En supposant que les données se trouvent dans le répertoire `DONNEES`, le post-traitement s'exécutera avec la commande :

```
python chemin_vers_bbob_pproc/run.py DONNEES
```

Ceci créera un dossier `ppdata` dans lequel se trouveront figures et tables.

L'atelier de recherche BBOB a permis de démontrer l'intérêt de la communauté scientifique pour le logiciel COCO.

Conclusion et perspectives

Ce manuscrit présente nos contributions dans les domaines de l'optimisation continue boîte noire et dans l'évolution artificielle. Nous avons tâché de répondre à la question de la comparaison d'algorithmes pour l'optimisation d'un problème boîte noire. En particulier, nous voulons savoir quel algorithme choisir pour minimiser le coût de l'optimisation.

Nous avons proposé et mis en pratique des méthodologies expérimentales pour la comparaison d'algorithmes sur des fonctions tests artificielles. Ces comparaisons ont permis d'obtenir des idées utiles du comportement d'algorithmes face à des difficultés rencontrées dans l'optimisation de problème du monde réel.

Face à des problème à grande dimensionalité, la complexité quadratique de certains algorithmes comme CMA-ES peut être prohibitive. Nous avons proposé deux variantes sep-CMA-ES et block-CMA-ES qui utilisent un modèle avec une complexité moindre : la matrice de covariance est restreinte à ses blocs diagonaux. En ajustant les paramètres de l'algorithme, nous montrons que les performances de ces variantes sont meilleures que celles de CMA-ES pour des problèmes séparables ou partiellement séparables.

Ces variantes constituent une preuve de concept pour une utilisation de l'adaptation de la matrice de covariance dans les problèmes à grande dimensionalité. Ces variantes ouvrent aussi la porte à de nouvelles stratégies où la complexité de la matrice de covariance pourrait évoluer au cours de l'optimisation du problème.

Par ailleurs, nous avons proposé deux études de comparaisons systématiques d'algorithmes d'optimisation. Nous avons ainsi montré que de simples rotations de l'espace de recherche pouvaient affecter les performances d'algorithmes comme PSO, NEWUOA et BFGS. Par ailleurs, NEWUOA ou BFGS sont affectés par la non-convexité de la fonction objectif.

Dans un deuxième temps, nous avons mené une étude à grande échelle sur un ensemble conséquent de fonctions tests et d'algorithmes dans le cadre de l'atelier de recherche BBOB. Nous avons montré que certains algorithmes comme le simplexe de Nelder-Mead obtient les meilleurs résultats en dimension 2 et 3 sur un ensemble de fonctions non bruitées mais que cela n'est pas vrai quand la dimension du problème est plus grande. Les performances dépendent aussi de la quantité d'évaluations de fonction allouée, en particulier pour un nombre important d'évaluations les techniques reposant sur l'adaptation de la matrice de covariance sont les meilleures, ceci pour des fonctions bruitées ou non. Les perspectives à BBOB et à l'utilisation de COCO sont nombreuses : l'intégration de fonctions du monde réel ou de problèmes à plus grande dimensionalité, l'extension de COCO à d'autres fonctions tests et d'autres types d'optimisations en sont des exemples.

Acknowledgements

Now this manuscript is almost complete, there are many people which I would like to acknowledge for their involvement in my work and my life during these years of preparation for the PhD.

First, I would like to thank Professor Bonnans for accepting to be a member of my jury, Professor Rudolph and Doctor Le Riche for reviewing my work and for their helpful comments. I will be very glad to meet you on December 21 and eventually afterward during the rest of my career.

I would like to acknowledge the staff of the Laboratoire de Recherche en Informatique whom I came across these years. You are what made the halls of the laboratory livelier every day.

Also part of the LRI, I would like to address a kind thought to the Doctoral School of Computer Science and its administration. Having been a student representative for one year, I know their dedication to turning PhD candidates into full-fledged researchers (sometimes against their own will).

Next, I would like to acknowledge my fellow researchers interns and fellow PhD candidates whom I have been acquainted with. Still some years to go!

Then there are the people of the TAO team-project. Among you I have found people that are more than colleagues.

I will start with you Steffen: though it was only for a few months, I am sure our collaboration on BBOB will keep paying off for a while. I wish you the best of luck for the end of your PhD.

Marie-Carol, working with you always makes our day brighter. Truly, many thanks to you.

Antoine, thanks for your insights and coping with me for this long.

Marc and Michele, I would like to thank you both for making all of our work hold

in one single ‘container’ and gathering the people of TAO under your supervision. The collaboration and work environment I have found in TAO is in many aspects thanks to both of you.

Anne, I have a particular thought for you. Through my friendship with Mohamed and other PhD candidates, you have been presented to me as a demanding superviser, talented and enthusiastic fellow researcher. My own experience has revealed that you were both and even more. I thank you for your concern for other people, me included.

Niko, I have already addressed many thanks to you throughout the process of writing this manuscript. Many things during the preparation of my PhD have been made possible only because of our collaboration and your supervision. Though many people must have done that before, I would like to acknowledge your work ethic, your scientific perceptiveness and wish it rubs off on me at some point.

I would like to gratefully acknowledge my relatives and friends to whom I have imposed my absence and unavailability. There are many things I should write down about you but I would rather say them to you directly. Among the people with whom I would like to make up for the lost time, I have a special thought for:
JB: Thanks for correcting my typos, Fabien: I hope to catch up with you at go, Rafael: up for some games?, Sylvain: Google is watching you, Olivier: congratulations for your soon-to-come child, Cédric: still your best man, Jean-Baptiste P. and Raphael: now we can get back to serious things, Michel: you will drive from now on, Mohamed: you made me realise I like sharing office AND ideas about everything.

Grand-father, you have supported me all this time. Thanks.

My last thought goes to my close family and Isabelle. You have continuously shown me your support, each in your own way. I know that you will keep doing so. I am where and who I am thanks to you. I love you.

To everyone that I have forgotten, a single word: thanks!

PhD Thesis

by

Raymond ROS

Real-Parameter Black-Box Optimisation:

Benchmarking and Designing Algorithms

Laboratoire de Recherche en Informatique, U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France

Abstract

In continuous optimisation a given problem can be stated as follows: given the objective function f from $\mathbb{R}^n$ to $\mathbb{R}$ with n the dimension of the problem, find a suitable vector that minimises f up to an arbitrary numerical precision. In this context, the black-box scenario assumes that no information but the evaluation of f is available to guide its optimisation.

In the first part, we study the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) which is a well-established stochastic approach for solving Black-Box Optimisation (BBO) problems. We show its time and space complexity limits when addressing high-dimensional BBO problems. To overcome such limits, we provide variants of the CMA-ES that update only block-diagonal elements of the covariance matrix, and exploit the separability of the problem. We show that on non-separable functions these variants can outperform the standard CMA-ES, given that the dimension of the problem is large enough.

In the second part, we define and exploit an experimental framework BBO Benchmarking (BBOB) in which practitioners of BBO can test and compare algorithms on function testbeds. Results show dependencies on the budget (number of function evaluations) assigned to the optimisation of the objective function. Some methods such as NEWUOA or BFGS are more appropriate for small budgets. The CMA-ES approach using restarts and a population size management policy performs well for larger budgets.

The COmparing Continuous Optimisers (COCO) software, used for the BBOB, is described technically in the third part. COCO implements our experimental framework as well as outputs the results that we have been exploiting.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

An optimisation problem can be formalised using mathematical notions: optimisation is looking for some element $\boldsymbol{x}_{\text{opt}}$ of the domain Ω a subset of $\mathbb{R}^n$, with respect to the *objective function* $f : \Omega \to \mathbb{R}$, such that $f(\boldsymbol{x}_{\text{opt}})$ is lesser than $f(\boldsymbol{x})$ for $\boldsymbol{x}$ in a subset of Ω. We described the specific case of minimisation of the objective function f; maximisation is equivalent without loss of generality to the minimisation of its opposite $-f$. The optimum would usually be a solution that is good enough according to some given quality criterion; for instance, instead of searching for the optimum, we can look for an optimal value of f up to an arbitrary precision. One can object that choosing such a solution $\boldsymbol{x}$ is a *multi-objective* optimisation problem since the quality criterion might be a compromise between multiple sub-criteria, but we choose not to discuss this matter and restrict our search to *single-objective* problems for which only the value of $f(\boldsymbol{x})$ is considered to evaluate the quality of a candidate solution $\boldsymbol{x}$.

Optimisation finds applications in many different fields such as aerodynamics with the conception of airfoils, biology with the calibration of models, electronics with circuit tuning, physics with molecular conformation of minimum energy for instance.

Our work is in the scope of *Continuous Optimisation* since the components of the solution vectors $\boldsymbol{x}$ are real values, as opposed to *Discrete Optimisation*. More specifically, we are considering *Unconstrained Optimisation* where no constraint has to be taken into account in the search of the optimum.

In continuous optimisation, the *black-box* scenario assumes that the only way to gather information on the problem at hand is to evaluate the objective function. For

instance, no assumption is made on the separability, the presence of noise, the availability of the gradient or the Hessian of the function. Though some real-world problems may provide such information, most problems are fundamentally black boxes, because no mathematical expression of the objective function is available or the function evaluation requires some physical simulation or experiment. Facing such Black-Box Optimisation (BBO) problems which come without prior knowledge, the choice of an appropriate optimisation algorithm is a difficult task.

Many algorithms using a deterministic approach or which are heuristics-based can be found in the literature from the fields of *Global Optimisation*, *Operational Research* or *Evolutionary Computation*.

In this context, our key question is: how can we compare algorithms in order to decide whether an algorithm might be appropriate for a given BBO problem?

There exists no theoretical solid ground on which to stand on to choose between different methods, because of the lack of information on the objective function considered. All theoretical results either make simplifying hypotheses that are not valid for real-world problems or give results that do not yield any outcome relevant to practice.

Systematic experimental procedures, sometimes coming with their software implementation, are a way of comparing algorithms, therefore they provide parts of an answer to our key question. We regroup these systematic procedures under the appellation 'benchmarking', a 'benchmark' being an instance of benchmarking. With regard to benchmarking, we are interested in the notion of *search costs* which we define as the quantity of computation that is required for an algorithm to reach a solution.

A rigourous benchmark implementation of BBO algorithms has been proposed [Hansen, 2006b, Suganthan et al., 2005] which compares the search costs of many different algorithms on a number of problems. However a critical issue is that of the collection of problems constituting the benchmark suite. Since no test suite of real-world problems can possibly cover the whole range of difficulties encountered in BBO, the choice has been made in [Suganthan et al., 2005] and in this thesis to make use of artificial test functions featuring a chosen set of properties that are known to make optimisation problems difficult. The benchmarking of algorithms on a test suite of artificial functions can help identifying the weaknesses and strong points of the algorithm with respect to these BBO difficulties. Nevertheless, the results of this

benchmarking should be generalised to other problems only with caution.

Let us motivate possible uses of benchmarking. When confronting a new problem, if partial information is available —which we can refer to as a 'grey-box' scenario—, the results of the benchmarking will provide candidate algorithms that are known to perform well with respect to the identified characteristics of the problem. Inversely, if the new problem is a real 'black-box' and no information is available, the results of a few chosen algorithms could very well provide a characterisation of the new problem in the light of the performances of these algorithms on the benchmark.

In the process of benchmarking, the 'curse of dimensionality' [Bellman, 1961] is one well-known difficulty of optimisation which refers to the exponential increase of the volume of the search space as the dimension increases. A consequence of the 'curse of dimensionality' is that a search policy which is appropriate when the dimension of the search space is small might be useless when the dimension is moderate or large. Therefore, the study of the *scalability* of algorithms is particularly relevant.

A number of benchmark implementations are available to the practitioner, each one of them answering the needs of a community of practitioners. The community of evolutionary computation found in the special session on real-parameter optimization CEC 2005 [Hansen, 2006b, Suganthan et al., 2005], an experimental framework aiming for the rigourous and fair comparison of algorithms as well as an implementation that proposes twenty-five problems coded in different programming languages. A limitation to benchmarks is that the number of algorithms tested depends on whether there exists an implementation of the algorithms compatible with the benchmark implementation. Many practical issues such as floating point exceptions or those encountered in the management of experimental data have to be taken into account for the implementation of a benchmark.

A review of the state of the art of BBO methods and benchmark implementations is given in the following chapter. The first part of this thesis focuses on the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES), and describes simple modifications of the algorithms that can deal with high dimensional problems. The second and third part of this thesis revolve around our benchmark experiments and more specifically the BBO Benchmarking 2009 and its software implementation COCO.

Nomenclature

Abbreviations

BBO Black-Box Optimisation

BBOB Black-Box Optimisation Benchmarking

CPU Central Processing Unit

ECDF Empirical Cumulative Distribution Function

ERT Expected Running Time

SP1 Success Performance One

Algorithms

For more details, see Sections 2.1 and 4.4.1.

ACO Ant Colony Optimization

BayEDA$_{\mathbf{cG}}$ Bayesian Estimation of Distribution Algorithm

BFGS Broyden Fletcher Goldfarb Shanno

DASA Differential Ant-Stigmergy Algorithm

DE Differential Evolution

EA Evolution Algorithm

ES Evolution Strategy

GA Genetic Algorithm

G3-PCX Generalized Generation Gap model with Parent Centric CROSSover

IDEA Iterated Density-estimation Evolutionary Algorithm

- **AMaLGaM IDEA** Adapted Maximum-Likelihood Gaussian Model Iterated Density-estimation Evolutionary Algorithm
- **iAMaLGaM IDEA** incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density-estimation Evolutionary Algorithm

NEWUOA NEW Unconstrained Optimization Algorithm

MA-LS-Chain Memetic Algorithm using Local Search Chaining

MCS Multilevel Coordinate Search

POEMS Prototype Optimization with Evolved IMprovement Steps

PRS Pure Random Search

PSO Particle Swarm Optimization

- **DEPSO** Differential Evolution Particle Swarm Optimization
- **EDA-PSO** Estimation of Distribution Algorithm Particle Swarm Optimization
- **PSO_Bounds** Particle Swarm Optimization with adaptive bound procedure

SNOBFIT Stable Noisy Optimization by Branch and FIT

VNS Variable Neighbourhood Search

Covariance Matrix Adaptation

CMA-ES Covariance Matrix Adaptation Evolution Strategy

- **BIPOP-CMA-ES** BI-Population Covariance Matrix Adaptation Evolution Strategy
- **block-CMA-ES** block Covariance Matrix Adaptation Evolution Strategy

CMA Covariance Matrix Adaptation

$(\mu/\mu_W, \lambda)$-CMA-ES Covariance Matrix Adaptation Evolution Strategy with weighted recombination of μ parents and λ offspring

IPOP-CMA-ES Increasing POPulation Covariance Matrix Adaptation Evolution Strategy

sep-CMA-ES separable Covariance Matrix Adaptation Evolution Strategy

MVA-ES Main Vector Adaptation Evolution Strategy

Software

See Section 2.2.1.

AMPL A Modeling Language for Mathematical Programming

API Application Programming Interface

COCO COmparison of Continuous Optimisers

COCONUT COntinuous CONstraint-Updating the Technology

COIN-OR COmputational INfrastructure for Operations Research

CUTEr Constrained and Unconstrained Testing Environment, revisited

DFO Derivative-Free Optimization

GAME Group of Adaptive Model Evolution

JCOOL Java COntinuous Optimization Library

OAT Optimization Algorithm Toolkit

Notations

$\circ : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$ operator of the element-wise multiplication of vectors

argmin argument of the minimum operator

$\boldsymbol{e}_j \in \mathbb{R}^n$ j-th coordinate vector

$\Delta f \in \mathbb{R}$ arbitrary precision

$f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ objective function to be minimised

$f_{\mathbf{opt}} = f(\boldsymbol{x}_{\mathbf{opt}}) \in \mathbb{R}$ optimal value of f

$f_{\text{target}} = f(\boldsymbol{x}_{\mathbf{opt}}) + \Delta f \in \mathbb{R}$ target function value

$\boldsymbol{I} \in \mathbb{R}^n$ identity matrix

$k \in \mathbb{N}$ denotes the step (k-th) of an algorithm

$\mathbb{N}^*$ is $\mathbb{N} \setminus \{0\}$

$n \in \mathbb{N}^*$ dimension of the search space

∇f gradient of f,

$\nabla^2 f$ Hessian matrix of f,

$\nabla^2 f^{-1}$ the inverse of the Hessian matrix of f,

$\boldsymbol{x}$ vector $\boldsymbol{x}$

$\boldsymbol{X}$ matrix $\boldsymbol{X}$

$\boldsymbol{x}_{\mathbf{opt}} \in \mathbb{R}^n$ optimum of f

BFGS

$\alpha^{(g)} \in \mathbb{R}$ step length

$\boldsymbol{p}^{(g)} \in \mathbb{R}^n$ descent direction

$\boldsymbol{B}^{(g)} \in \mathbb{R}^{n \times n}$ Hessian approximate

NEWUOA

$c^{(g)} \in R$, $\boldsymbol{g}^{(g)} \in \mathbb{R}^n$, $\boldsymbol{G}^{(g)} \in \mathbb{R}^{n \times n}$ parameters of the model $m^{(g)}$

$m^{(g)} : \mathbb{R}^n \mapsto \mathbb{R}$ quadratic model

$\boldsymbol{p}^{(g)} \in \mathbb{R}^n$ iteration step

$q \in \mathbb{N}^*$ number of interpolation points

$\rho^{(g)} \in \mathbb{R}$ step acceptance ratio

PSO

$\boldsymbol{g} \in \mathbb{R}^n$ global best position

$i \in \{0, \ldots, S\}$ denotes the i-th particle

$K \in \mathbb{N}$ number of neighbours considered to determine $\boldsymbol{n}_i$

$\boldsymbol{n}_i \in \mathbb{R}^n$ best neighbour of the i-th particle

$\boldsymbol{p}_i \in \mathbb{R}^n$ previous best position of the i-th particle

$\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3 \in \mathbb{R}^n$ random unit vectors

S swarm size, population size

$\boldsymbol{v}_i \in \mathbb{R}^n$ velocity of the i-th particle

$w, c_1, c_2, c_3 \in \mathbb{R}$ algorithm constants

$\boldsymbol{x}_i \in \mathbb{R}^n$ position of the i-th particle

DE

F weighting factor

CR crossover factor

$\boldsymbol{x}_{\mathbf{best}}$ best vector so far

ES

$\boldsymbol{C} \in \mathbb{R}^{n \times n}$ covariance matrix

$\lambda \in \mathbb{N}^*$ population size, number of offspring

$\mu \in \mathbb{N}^*$ number of parents

$(\mu\{, +\}\lambda) - ES$ selection operator to occur within the λ offpsring, the $\mu + \lambda$ individuals respectively for the comma and plus operator

$\mathcal{N}(0, \boldsymbol{I})$ multi-variate normal distribution with zero mean and unity covariance matrix

$\mathcal{N}(\boldsymbol{m}, \boldsymbol{C}) \sim \boldsymbol{m} + \mathcal{N}(\boldsymbol{m}, \boldsymbol{C})$ multi-variate normal distribution with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ symmetric and positive definite

$\sigma \in \mathbb{R}^+$ step size

$\sim$ equality operator for distributions

CMA-ES

$\boldsymbol{B}^{(g)} \in \mathbb{R}^{n \times n}$ an orthogonal matrix which columns are eigenvectors of $\boldsymbol{C}^{(g)}$ and correspond to the diagonal elements of $\boldsymbol{D}^{(g)}$

$\boldsymbol{C}^{(g)} \in \mathbb{R}^{n \times n}$ covariance matrix at generation g

$c_{jj}^{(g)} \in \mathbb{R}$ diagonal element of $\boldsymbol{C}$

$c_{\mathbf{c}} \in [0, 1]$ learning rate for the cumulation for the rank-one update of the covariance matrix

$c_{\mathbf{cov}} \in [0, 1]$ learning rate for the covariance matrix update

$c_\sigma \in]0, 1]$ learning rate for the cumulation for the step size control

$\boldsymbol{D}^{(g)} \in \mathbb{R}^{n \times n}$ a diagonal matrix which elements are the square root of eigenvalues of $\boldsymbol{C}^{(g)}$ and correspond to the columns of $\boldsymbol{B}^{(g)}$

$d_\sigma \approx 1$ **damping parameter for step size update**

E expectation value

$\mu_{\mathbf{cov}}$ parameter for weighting between rank-one and rank-μ update

$\mu_{\mathbf{eff}} = (\sum_i^\mu = 1 w_i^2)^{-1}$ variance effective selection mass

$\boldsymbol{p}^{(g)} \in \mathbb{R}^n$ evolution path at generation g, a sequence of successive (normalised) steps

$(\boldsymbol{p}^g)_j \in \mathbb{R}$ j-th coordinate of $\boldsymbol{p}^{(g)}$

$\sigma^{(g)} \in \mathbb{R}^+$ step size at generation g

$\langle \boldsymbol{x} \rangle_{\mathbf{W}}^{(g)} = \sum_{i=1}^\mu w_i \boldsymbol{x}_{i:\lambda}^{(g)} \in \mathbb{R}^n$ weighted mean of the individuals $\boldsymbol{x}_{i:\lambda}^{(g)}$

$\boldsymbol{x}_{i:\lambda}^{(g)} \in \mathbb{R}^n$ the i-th best, ranked according to objective function f, out of the λ individuals $\boldsymbol{x}_k^{(g)}$ at generation g

$w_i \in \mathbb{R}$ with $i = 1, \ldots, \mu$ recombination weights

$\langle \boldsymbol{z} \rangle_{\mathbf{W}}^{(g)} = \sum_{i=1}^\mu w_i \boldsymbol{z}_{i:\lambda}^{(g)} \in \mathbb{R}^n$ weighted mean of the individuals $\boldsymbol{z}_{i:\lambda}^{(g)}$

$\boldsymbol{z}_k^{(g+1)} \in \mathbb{R}^n$ are normally ($\mathcal{N}(0, \boldsymbol{I})$) distributed vectors

$\boldsymbol{z}_{i:\lambda}^{(g)} \in \mathbb{R}^n$ the i-th best, ranked according to objective function f, out of the λ individuals $\boldsymbol{z}_k^{(g)}$ at generation g

$(\boldsymbol{z}_{i:\lambda}^{(g)})_j \in \mathbb{R}$

block-CMA-ES

$\boldsymbol{B}_j^{(g)}$, $\boldsymbol{C}_j^{(g)}$, $\boldsymbol{D}_j^{(g)}$ j-th block of $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ respectively

$(p_{\mathbf{c}}^{(g+1)})_j$, $(\boldsymbol{z}_{i:\lambda}^{(g)})_j \in \mathbb{R}$ projections of $p_{\mathrm{c}}^{(g+1)}$) and $\boldsymbol{z}_{i:\lambda}^{(g)}$ onto the j-th subspace, different from the same notations in the context of CMA-ES

Test Functions

$\boldsymbol{Q}$ is either $\boldsymbol{I}$ or an orthogonal $n \times n$ matrix with each column vector $\boldsymbol{q}_i$ being a uniformly distributed unit vector implementing an angle-preserving transformation.

β condition number/parameter

Performance Measurements

Let us consider that we have run multiple trials on a given problem. A trial is considered successful if the target function value f_{target} is reached. We will consider for a successful run the number of function evaluations for reaching the target function value which we call *running time*. We can define:

$p_{\mathbf{S}}$ the probability of success, the ratio of the number of successful runs over the total number of runs

$\mathbf{RT_S}$ is the average of the running times of the successful runs.

$\mathbf{RT_{US}}$ is the average of the running times of the unsuccessful runs.

$SP1 = \frac{\mathbf{RT_S}}{p_{\mathbf{S}}}$ the success performance one

$\text{ERT} = \frac{p_{\mathbf{S}}\mathbf{RT_S}+(1-p_{\mathbf{S}})\mathbf{RT_{US}}}{p_{\mathbf{S}}}$ the expected running time

Chapter 2

Review of the State of the Art

Contents

In the face of a BBO problem, deciding which method is appropriate is a difficult task, especially since so many methods from the literature in Global Optimisation, Operational Research and Evolutionary Computation are available.

In the field of operational research, some methods are appropriate for solving BBO problems. These methods are called 'derivative-free' optimisers because they can proceed without the availability of the gradient of the objective functions. The appellation 'derivative-free optimization' has been used with some frequency in the community of model-based algorithms [Conn and Toint, 1996, Conn et al., 1997a].

Global optimisation lists exact methods and heuristics, denoted as 'direct search' methods which could also be used to solve BBO problems. The appellation 'direct search' is introduced in [Hooke and Jeeves, 1961] in the context of pattern search methods and is more properly defined in [Kolda et al., 2003] which also mentions 'derivative-free' optimisation. Among these direct search methods, there are for instance 'globalised' extensions of local search methods from applied mathematics which we discuss further down.

Though it is not in the scope of our work to find a way to conciliate global optimisation, operational research and evolutionary computation, we would like to initiate a move in the direction of a reunified theoretical and experimental framework through our work.

Benchmarks of algorithms through systematic experimental procedures provide parts of an answer to the key question of this thesis. Many implementations are

out there within range of the BBO practitioner. These software implementations are often only known by people from the field these implementations originated from.

Section 2.1 briefly covers the main types of BBO methods available to the practitioner. Section 2.2 provides descriptions of the existing software facilities for the practitioner in BBO to experiment with.

2.1 Types of Solvers

In the field of BBO, many different approaches exist, providing different solvers or algorithms that strive to solve the BBO problem considered. From now on, we will use the words 'solver' or 'optimisation algorithm' without distinction. Usually these solvers would be an application of a theoretical idea or would be designed in order to solve a specific problem and generalised to others.

These solvers are iterative: they iteratively try to improve from an initial guess to try and reach the global optimum. The algorithms we briefly present here can be split into two categories: the ones that have a deterministic behaviour and the ones that base their functioning on stochasticity.

From this point on, we will refer without distinction to optimisation and minimisation since we can account for maximisation by considering the opposite of the objective function f without loss of generality.

2.1.1 Deterministic Methods

In this section, we present deterministic methods [Nocedal and Wright, 2006] used in unconstrained continuous optimisation. Two dominant classes are the line search and the trust region methods presented in Sections 2.1.1.1 and 2.1.1.2. Another class of methods are pattern search methods which we present in Section 2.1.1.3. The methods we mentioned are appropriate for the search of local optimum, we describe in Section 2.1.1.4 how these methods can be adapted to global optimisation.

2.1.1.1 Line Search

In the case of line search methods, from the current iterate $\boldsymbol{x}^{(g)}$, the algorithm finds a direction of descent $\boldsymbol{p}^{(g)}$ and then the descent step length $\alpha^{(g)}$ from $\boldsymbol{x}^{(g)}$ to obtain the new iterate $\boldsymbol{x}^{(g+1)}$ is estimated by the following equation:

$$\alpha^{(g)} = \underset{\alpha>0}{\operatorname{argmin}} f(\boldsymbol{x}^{(g)} + \alpha \boldsymbol{p}^{(g)})$$
$$\boldsymbol{x}^{(g+1)} = f(\boldsymbol{x}^{(g)} + \alpha^{(g)} \boldsymbol{p}^{(g)})$$

Most often though, line search refers to the determination of the descent step given a direction of descent.

Among the descent directions available, the *steepest descent* given by the opposite of the gradient is an obvious choice resulting in the steepest descent method:

$$\boldsymbol{p}^{(g)} = -\nabla f(\boldsymbol{x}^{(g)})$$

The gluttonous behaviour of such method can lead to slow convergence on certain types of problems.

The *Newton* direction is given by the second derivative of the objective function:

$$\boldsymbol{p}^{(g)} = -\nabla^2 f^{-1}(\boldsymbol{x}^{(g)}) \nabla f(\boldsymbol{x}^{(g)})$$

This provides a reliable descent direction as long as the current iterate $\boldsymbol{x}^{(g)}$ is reasonably close to the global optimum of the objective function and the Hessian is positive symmetric definite. The Newton methods using the direction of the same name ensure convergence rate close to quadratic but need the computation of the Hessian of the function which can be expensive.

Quasi-Newton methods uses the following descent direction:

$$\boldsymbol{p}^{(g)} = -{\boldsymbol{B}^{(g)}}^{-1} \nabla f(\boldsymbol{x}^{(g)})$$

where the symmetric and positive definite matrix $\boldsymbol{B}^{(g)}$ is updated at each iteration using a formula. Different update formulae exist, resulting in different quasi-Newton methods such as the SR1 [Broyden, 1965, Fletcher, 1987] or the BFGS [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970].

The line search methods involve gradient computations, thus replaced with finite differences in the case of BBO. The gradient of f at $\boldsymbol{x}^{(g)}$ is approximated by evaluating f at $n+1$ points in the case of one-sided differences:

$$\frac{\partial f}{\partial x_j}(\boldsymbol{x}^{(g)}) \approx \frac{f(\boldsymbol{x}^{(g)} + \epsilon\, \boldsymbol{e}_j) - f(\boldsymbol{x}^{(g)})}{\epsilon}$$

along the j-th component, where $\boldsymbol{e}_j$ is the j-th unit vector and ϵ is a positive real number. To ensure some precision in the approximation of the gradient, ϵ needs to be small compared to the components of $\boldsymbol{x}^{(g)}$. In the case of floating point operations though, to prevent numerical round-off errors, ϵ cannot be arbitrarily small.

The *implicit filtering* method [Gilmore and Kelley, 1995] is, in its simplest form, a variant of the steepest descent method with line search adapted to optimisation with noise. The implicit filtering method works best on functions for which the noise level decreases as the iterates approach the optimum. The method uses a finite difference computation with a decreasing sequence $(\epsilon^{(g)})_{g\in\mathbb{N}}$ for ϵ. For each different value $\epsilon^{(g)}$, the descent direction $-\nabla_{\epsilon^{(g)}} f$ is obtained using the Armijo line search which consists in incrementing the integer m until the following equation is verified:

$$f(\boldsymbol{x}^{(g)} - \rho^m \nabla_{\epsilon^{(g)}} f(\boldsymbol{x}^{(g)})) \leq f(\boldsymbol{x}^{(g)}) - c\rho^m \|\nabla_{\epsilon^{(g)}} f(\boldsymbol{x}^{(g)}))\|_2^{\,2}$$

with $\rho \in [0,1]$. If m is larger than the constant $a_{\max}$, we use the next element of the decreasing sequence $(\epsilon^{(g)})_{g\in\mathbb{N}}$ instead. The implicit filtering method uses such strategy instead of choosing ϵ to ensure some precision on the gradient estimate which may be thrown off due to the noise. An improvement to the standard implicit filtering method [Kelley, 1999] consists in using the gradient estimate $\nabla_{\epsilon^{(g)}} f$ to construct approximate Hessians and thus generating quasi-Newton search directions.

Line search methods are essentially local search approaches. Some approaches, grouped under the appellation 'derivative-free nonmonotone' techniques, adapts line search to global search by generalising the Armijo rule to allow increase of the function value [Diniz-Ehrhardt et al., 2008, Grippo et al., 1986]. An efficient procedure for choosing the successive descent direction using this line search is still not determined.

2.1.1.2 Trust Region

Trust region methods use a model $m^{(g)}$ that is approaching the behaviour of the objective function around $\boldsymbol{x}^{(g)}$ in the trust region. This model has to verify some interpolation conditions: $m^{(g)}(\boldsymbol{y}_i) = f(\boldsymbol{y}_i)$ for $i = 1, \ldots, q$ with q the number of interpolation points. The interpolation set Y is the list of the vectors $(\boldsymbol{y}_i)$.

The step $\boldsymbol{p}^{(g)}$ is then determined by:

$$\boldsymbol{p}^{(g)} = \underset{\boldsymbol{p}}{\operatorname{argmin}}(m^{(g)}(\boldsymbol{x}^{(g)} + \boldsymbol{p}))$$

with the constraint that the norm of $\boldsymbol{p}^{(g)}$ must be less than $\Delta^{(g)} > 0$, the trust region radius. The steps are chosen to stay within the trust region. As the iterate $\boldsymbol{x}^{(g)}$ closes on the optimum, the trust region radius $\Delta^{(g)}$ should decrease.

The step acceptance and trust-region update strategies are based on the ratio between the actual reduction in the function and the reduction predicted by the model:

$$\rho^{(g)} = \frac{f(\boldsymbol{x}^{(g)}) - f(\boldsymbol{x}^{(g)} + \boldsymbol{p}^{(g)})}{m^{(g)}(\boldsymbol{x}^{(g)}) - m^{(g)}(\boldsymbol{x}^{(g)} + \boldsymbol{p}^{(g)})}$$

Since $\boldsymbol{x}^{(g)} + \boldsymbol{p}^{(g)}$ minimises $m^{(g)}$, the denominator is always strictly larger than zero. Hence, if $\rho^{(g)}$ is negative, $f(\boldsymbol{x}^{(g)} + \boldsymbol{p}^{(g)})$ is larger than $f(\boldsymbol{x}^{(g)})$ and the step is rejected. If $\rho^{(g)}$ is close to one, there is good agreement between the model and the objective function in the trust region therefore the trust region radius $\Delta^{(g)}$ can be increased. Otherwise if $\rho^{(g)}$ is positive but not so close to one, we may keep $\Delta^{(g)}$ or reduce its value depending on whether the interpolation set Y needs to be improved or not. The next iterate $\boldsymbol{x}^{(g+1)}$ is assigned to $\boldsymbol{x}^{(g)}$ or $\boldsymbol{x} + \boldsymbol{p}^{(g)}$ depending on whether $\rho^{(g)}$ is larger than η which is a constant in $[0, 1]$.

The model, if quadratic, can be written as:

$$m^{(g)}(\boldsymbol{x}^{(g)} + \boldsymbol{p}) = c^{(g)} + \boldsymbol{g}^{(g)^T}\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{G}^{(g)}\boldsymbol{p}$$

where $c^{(g)}$, $\boldsymbol{g}^{(g)}$ and $\boldsymbol{G}^{(g)}$ are parameters of the model $m^{(g)}$ which have overall $\frac{1}{2}(n + 1)(n+2)$ coefficients. The model can be fully interpolated by solving the linear system given by the interpolation conditions if the algorithm uses $\frac{1}{2}(n+1)(n+2)$ interpolation points, provided that the linear system is non-singular.

For more information, [Conn et al., 2000] proposes an exhaustive treatment of the state of the art in trust region methods.

2.1.1.3 Pattern Search

The pattern search algorithm originally refers to the algorithm described by Hooke and Jeeves [1961]. The 'pattern search' appellation has been generalised to algorithms that use a set of vertices around the current iterate $\boldsymbol{x}^{(g)}$ to direct the search of the optimum. The set of vertices may evolve as the search proceeds.

An instance of pattern search method is the coordinate search algorithm which consists in cycling through the n coordinate directions given by the unit vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ and obtaining new iterates by performing line search along each direction in turn. The coordinate search can iterate infinitely but provides with an alternative that does not require the computation of the gradient estimate. The speed of convergence can be quite acceptable if the variables of the objective function f are loosely coupled. The original pattern search algorithm [Hooke and Jeeves, 1961] actually is a coordinate search that performs the sequence of coordinate descent and then search along the joint line between the first and last points in the cycle.

Pattern search methods are a generalisation of the coordinate search in the sense that they allow to search in a richer set of directions at each iterations.

The *downhill simplex method* [Lagarias et al., 1998], introduced by Nelder and Mead [1965], is an instance of pattern search method. The downhill simplex method evolves a hull of $n+1$ points. The points of the hull satisfy the non-degeneracy condition which consists in preventing that the volume of the hull is zero. Given the list of vertices : $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ sorted from the best to the worst, we compute the centroid $\boldsymbol{c}$ of all but the worst vertex: $\boldsymbol{c} = \sum_{i=1}^{n} \boldsymbol{x}_i$. The worst vertex $\boldsymbol{x}_{n+1}$ is replaced by a new vertex using reflection, expansion or contraction. In the case where an improvement cannot be made, a reduction step occurs.

We describe these different operations below:

Reflection The reflection point is obtained using the following equation $\boldsymbol{x}_r = \boldsymbol{c} + \alpha(\boldsymbol{c} - \boldsymbol{x}_{n+1})$, the standard value for α is 1. Depending on the value of the reflection point compared to that of the best and second worst vertex, respectively $\boldsymbol{x}_1$ and $\boldsymbol{x}_n$, we can either decide to keep the reflection point, proceed with the expansion or the contraction transformations. If the value of the reflection point

is between that of the $\boldsymbol{x}_1$ and $\boldsymbol{x}_n$, then we replace the worst vertex $\boldsymbol{x}_{n+1}$ with that of the reflection point $\boldsymbol{x}_r$ and start a new iteration. Otherwise we proceed to the next transformation.

Expansion If the value of the reflection point $\boldsymbol{x}_r$ is better than that of the best point $\boldsymbol{x}_1$, the expansion point is computed: $\boldsymbol{x}_e = \boldsymbol{c} + \gamma(\boldsymbol{x}_r - \boldsymbol{c})$, the standard value of γ is 2. If the value of the expansion point is better than that of $\boldsymbol{x}_r$, then $\boldsymbol{x}_{n+1}$ is replaced by the expansion point $\boldsymbol{x}_e$, else by the reflection point. Then we start a new iteration.

Contraction If the value of the reflection point is worst than that of $\boldsymbol{x}_n$, the contraction point is computed.

Outside Contraction If the value of the reflection point is not worse than that of $\boldsymbol{x}_{n+1}$, the contraction point is computed as follows: $\boldsymbol{x}_c = \boldsymbol{c} + \beta(\boldsymbol{x}_r - \boldsymbol{c})$, the standard value of β is $\frac{1}{2}$. In this case, if the value of the contraction point is better than that of $\boldsymbol{x}^r$, $\boldsymbol{x}_{n+1}$ is replaced by the contraction point and a new iteration is started. Otherwise, we proceed to the shrink transformation.

Inside Contraction If the value of the reflection point is worse than that of $\boldsymbol{x}_{n+1}$, the contraction point is obtained as follows: $\boldsymbol{x}_c = \boldsymbol{c} + \beta(\boldsymbol{x}_1 - \boldsymbol{c})$. In this case, if the value of the contraction point is better than that of $\boldsymbol{x}_1$, $\boldsymbol{x}_{n+1}$ is replaced by the contraction point and a new iteration is started. Otherwise, we proceed to the shrink transformation.

Shrink All but the best point are replaced by: $\boldsymbol{x}_1 + \delta(\boldsymbol{x}_i - \boldsymbol{x}_1)$ with $i = 2, \ldots, n+1$, the standard value of δ is $\frac{1}{2}$. A new iteration is then started.

The Nelder-Mead algorithm is shown to fail in attaining an optimum [McKinnon, 1998] on a family of strictly convex objective functions in $\mathbb{R}^2$ by degeneration of the simplex in a sub-space. This effect is dependent on the initialisation of the algorithm. Restarts of the algorithms is often advocated as the easiest fix.

2.1.1.4 Deterministic Methods for Global Optimisation

The task of global optimisation is challenging [Dixon et al., 1976, Horst, 2002], Pintér [1996] gives an overview of methods for global optimization that covers a wider spectrum than that of BBO.

In the case of bound-constrained optimisation problems, the literature in global optimisation provides with methods that iteratively partition the search space and decompose the global problem into many local sub-problems, see for instance [Huyer and Neumaier, 1999, Jones et al., 1993]. These methods originates from the Branch-and-Bound method of combinatorial optimisation, see for instance [Nemhauser and Wolsey, 1988].

In the case of BBO where no information is available, some heuristics can be used to augment deterministic local search approaches addressing global optimisation. One such heuristic called multi-start consists in starting —and restarting— a local search algorithm from several points of the search space. This increases the probability of the global convergence of the local search process. If we consider only the starting points of the multi-starts, the process is the same as the Monte Carlo search, see Section 2.1.2.1. The inefficiency of the multi-start is two-fold: 1. the Monte Carlo search is clearly inefficient especially in the face of the 'curse of the dimensionality': the search space volume increases exponentially with n, making space filling sampling intractable even for moderate dimensionalities; 2. the probability of a single start of the local search process falling into a local optimum is left unchanged.

Extensions to the multi-start exist such as: 1. clustering relates to the search space partitioning methods mentioned since it consists in grouping the sampled points depending on the local optimum attained by the local search procedure to decide which area of the search space to sample at the next iteration [Törn and Žilinskas, 1989], 2. quasi-random sampling uses quasi-random sequence to sample more 'uniformly' the space, see for instance [Kucherenko, 2006]. Both extensions try to improve the quality of the sampling of the search space.

2.1.2 Stochastic Methods

In contrast with the previous deterministic approaches, Monte Carlo methods based on stochastic sampling of the search space, are presented here. The field of *Evolutionary Computation* provide with a number of stochastic techniques including evolutionary algorithms and swarm intelligence approaches. We describe some of these algorithms.

2.1.2.1 Monte Carlo Search

The Pure Random Search (PRS) [Brooks, 1958], also referred to as Monte Carlo search is the most simple stochastic search algorithm which consists in sampling each search point independently using a fixed probability distribution in the search domain and keeping the best solution found. It has been theoretically proven that PRS converges to the global optimum with probability one for any objective function given that the neighbourhood of the optimum can be sampled with a strictly positive probability. In practice, such approach comes with very large convergence time increasing exponentially with the dimension of the search space [Zhigljavsky and Žilinskas, 2008].

2.1.2.2 Simulated Annealing

Annealing refers to a technique in metallurgy which consists in heating and then controlling the process of cooling of a material to improve its overall physical properties. The heating causes the atoms to leave their initial configuration, which is a local optimum of energy. The controlled cooling increases the chance of having the atoms in a state of energy lower than initially. Simulated Annealing is a meta-heuristic for global optimisation that is conceptually analogous to annealing, with elements $\boldsymbol{x}$ of the search space being the different states of some physical system and the objective function providing the energy of these states. Simulated Annealing seeks to find a state of lower energy, in other words to improve a potential solution, by choosing a neighbour solution with a probability increasing with the closeness of the two solutions and decreasing with the value of a parameter which is denoted as the temperature. Originally developed for combinatorial optimisation [Kirkpatrick et al., 1983], the simulated annealing has been adapted to continuous optimisation [Wang and Chen, 1996].

In practice, the sequence of temperatures is essential to solve optimisation problems, this fact is stressed in [Spall, 2003] which gives an overview of the simulated annealing for global optimisation. The sequence needs to be decreasing to zero so that we reach an optimum but the decrease needs to be slow enough to escape local optima.

2.1.2.3 Ant Colony Optimisation

Ant Colony Optimisation (ACO) [Bilchev and Parmee, 1995, Colorni et al., 1991] regroups algorithms belonging to the field of evolutionary computation which uses the biological image of ants seeking paths from the base location of their colony to sources a food. ACO is based on the theory of *stigmergy*, which is another appellation for the ant colony paradigm and interprets the ability of a colony to interact through its environment.

Originally adapted to combinatorial optimisation problems, ACO can be extended to BBO either by using discretisation or by the use of probability sampling of the search space, see for instance [Socha and Dorigo, 2008].

2.1.2.4 Particle Swarm Optimisation

Particle Swarm Optimization (PSO) is a subset of evolutionary computation. The PSO algorithm [Clerc and Kennedy, 2002, Kennedy and Eberhart, 1995, Shi and Eberhart, 1998, Shi et al., 1999] is based on the biological paradigm of a swarm of particles that 'fly' over the objective landscape, exchanging information about the best solutions they have 'seen'. More precisely, each particle updates its velocity, stochastically twisting it toward the direction of the best solutions seen by 1. itself and 2. some parts of the whole swarm; it then updates its position according to its velocity and computes the new value of the objective function. The canonical expression of the velocity and position update of the swarm can be written as follows:

$$\begin{aligned}\boldsymbol{v}_i &\leftarrow w\boldsymbol{v}_i + c_1\boldsymbol{r}_1 \circ (\boldsymbol{g}_i - \boldsymbol{x}_i) + c_2\boldsymbol{r}_2 \circ (\boldsymbol{p}_i - \boldsymbol{x}_i) + c_3\boldsymbol{r}_3 \circ (\boldsymbol{n}_i - \boldsymbol{x}_i)\\ \boldsymbol{x}_i &\leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i\end{aligned}$$

where $\boldsymbol{g}_i$ is the position of the "global best" particle, the $\boldsymbol{p}_i$ is the best position ever obtained for the i-th particle, $\boldsymbol{n}_i$ is the neighbourhood best obtained from the

subset of neighbours of the i-th particle in the swarm, the real numbers w, c_1, c_2, c_3 are constants, $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ are random unit vectors and $\circ$ is the element-wise multiplication[1].

There are many variants of the PSO which has led to the implementation of Standard-PSOs[2].

2.1.2.5 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are another subset of evolutionary computation techniques in the sense that EAs are bio-inspired optimisation algorithms which evolve a population of solutions. They are modelled after Darwin's theory of natural evolution. More precisely, EAs implement the idea that the emergence of species that are adapted to their environment results from the synergy between natural selection (survival of the fittest) and blind variations (random mutation of the genetic material from parents to offspring, independently of any adaptation). The analogy with biology is also retrieved in the terminology used for EAs. The *fitness* designates the objective function, the *population of individuals* denotes the set of possible solutions to the problem at hand. A *generation* which denote an iteration of the algorithm consists in different steps repeated until a termination criteria is reached. After the initialisation of the population, a generation can be described as follows:

Fitness Scoring is the evaluation of the individuals according to the fitness, usually EAs would stop at this step after some individuals with sufficiently good fitness were found,

Selection is the process of choosing *parents*, that means individuals among the population according to their score biased towards the ones with the best score, thus implementing a step of natural selection,

Variations regroup the operators used for generating *offspring* from the parents; these variations are denoted *mutations* and *crossovers* or *recombinations* which are respectively unary and k-ary operators.

[1] Please note that $\boldsymbol{a} \circ \boldsymbol{b}$ is equivalent to $\boldsymbol{a} \cdot \boldsymbol{b}^T$, with $\boldsymbol{a}$ and $\boldsymbol{b}$ being column vectors.

[2] Two subsequent standards, 2006 and 2007, have been implemented, both available at: http://www.particleswarm.info/Programs.html

EAs cover a spectrum of fields of applications much larger than that of BBO only, since all that matters is to find an effective representation of the problem at hand, which usually contributes for a large part to the success of EAs methods. This comes with its downside since an effective representation may not be one that can be easily interpreted and therefore improved with the comprehension of the problem.

EAs, just like evolutionary computation, exist in the form of multiple techniques and approaches including genetic algorithms and evolution strategies or estimation of distribution algorithms.

2.1.2.6 Genetic Algorithms

Genetic Algorithms (GAs) were introduced by Holland [1975] and designed initially to handle bit-string representation of problems. GAs has been extended to handle BBO problems by using a binary bit-string representation of elements of $\mathbb{R}^n$. The major drawback in using such representation is the explosion in the length of the bit-strings as n grows larger.

2.1.2.7 Evolution Strategies

Evolution Strategies (ESs) are another kind of evolutionary algorithms, introduced by Rechenberg [1973b], Schwefel [1981]. Following the generic scheme of EAs, the steps in the ESs are defined as follows: the selection step consists in choosing individuals based on their ranking in terms of fitness values, the variation step consists mainly in Gaussian mutation.

The selection step based on the ranking of the individuals according to their fitness value can either be among the offspring only, or among the population of offspring and parents.

The Gaussian mutation consists in generating from a parent $\boldsymbol{x}$ the offspring $\boldsymbol{y}$ as follows: $\boldsymbol{y} = \boldsymbol{x} + \sigma\mathcal{N}(0, \boldsymbol{C})$, where $\sigma\mathcal{N}(0, \boldsymbol{C}) = \mathcal{N}(0, \sigma^2\boldsymbol{C})$ denotes the multi-variate normal distribution with mean 0 and covariance matrix $\boldsymbol{C}$.

The population size is historically denoted as μ and the number of offspring as λ. Originally, ESs considered population size of one. In the case where μ is larger than one, another variation operator is involved: the linear recombination of μ parents is used to generate the λ offspring. So in this case, we have $\boldsymbol{y} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_i + \sigma\mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$, where the $(w_i)_{i\in\{1,\dots,\mu\}}$ are real values and the $\boldsymbol{x}_i$ are sorted according to their fitness

values. The intermediate recombination denotes the case where: $w_i = \frac{1}{\mu}$ for $i \in 1, \ldots, \mu$.

A standard notation to denote ESs is in the form (μ, λ) and $(\mu + \lambda)$, where the comma operator ',' means selecting among the λ offspring only and the plus operator '+' means selecting between the λ offspring *and* the μ parents.

An asset of EAs and particularly of ESs is the possibility of including strategy parameters such as σ, the step size, and the covariance matrix $\boldsymbol{C}$ in the loop of the evolution, therefore *adapting* the parameters in the process of the optimisation.

In the most basic version of adaptation, the one-fifth success rule, first proposed by Schumer and Steiglitz [1968] and discovered independently by Devroye [1972], Rechenberg [1973a], adapts the step-size by computing the empirical success probability and compare it to one fifth. The step size is multiplied or divided by a factor of $e^{1/3} \approx 1.4$, if the empirical success probability, that is the fraction of offspring which improved in terms of objective function value over the parents, is greater or lesser than one fifth. The one-fifth success rule is theoretically justified by results on the sphere and corridor function in the asymptotic case where n goes to infinity.

Many other adaptation schemes exist such as SA-ES, introduced by [Schwefel, 1981], which adapts the step size, or CMA-ES which adapts both the step size and the covariance matrix using notions called cumulation and path length control, see Chapter 3.

2.1.2.8 Differential Evolution

Differential Evolution (DE) [Price et al., 2005, Price, 1996, Storn, 1996, Storn and Price, 1995, 1997] is an evolutionary algorithm that uses a differential mutation procedure that consists in the addition of the weighted difference of two population vectors to a third vector. So when considering the traditional evolutionary loop, DE uses these steps:

Mutation $\boldsymbol{v}_i = \boldsymbol{x}_{i_1} + F(\boldsymbol{x}_{i_2} - \boldsymbol{x}_{i_3})$, where i is in $1, \ldots, NP$ with NP being the population size, i_1, i_2, i_3 are indices chosen in $1, \ldots, NP$ and F is a constant in the range $]0, 2]$,

Crossover $\boldsymbol{u}_i$ is the resulting individual of the crossover between the parent $\boldsymbol{x}_i$ and

the mutant candidate $\boldsymbol{v}_i$ which is generated by choosing component by component between those of $\boldsymbol{v}_i$ and $\boldsymbol{x}_i$ with probability CR and $1 - CR$ respectively, with the exception that one random component of $\boldsymbol{u}_i$ must correspond to that of $\boldsymbol{v}_i$,

Selection the candidate $\boldsymbol{u}_i$ replaces $\boldsymbol{x}_i$ if there is an improvement.

2.1.2.9 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) [Mühlenbein and Paaß, 1996], also named Probabilistic Model-Building Genetic Algorithms (PMBGAs) [Pelikan et al., 2002], or Iterated Density Estimation Algorithms (IDEAs) [Bosman and Thierens, 2000], are inspired from genetic algorithms; instead of maintaining a population of individuals to represent potential solution vectors, EDAs use a probability distribution. The EDAs iteratively identify the parameters of the probability distribution by successively: 1. sampling the distribution using the current values of the parameters, 2. computing the fitness of the sampled points, 3. selecting some of these points with a bias towards the best point, 4. and either reconstructing a probability distribution from these points or updating the current distribution. EDAs originally address problems with bit-string representation but have been adapted to continuous optimisation, for instance in [Bosman and Thierens, 2000, Sebag and Ducoulombier, 1998]. A survey of EDAs can be found in [Larrañaga and Lozano, 2001, Pelikan et al., 2002]. EDAs and the Covariance Matrix Adaptation (CMA), which adapts a multi-variate normal search distribution, are shown to be related in many aspects though some key differences exist [Hansen, 2006a].

2.2 BBO Benchmarking Software

Facing Black-Box Optimisation (BBO) problem which comes without prior knowledge, choosing an optimisation algorithm is a difficult task. A way of providing decisive elements to the choice process is the theoretical analysis of the convergence rates of the algorithms considered, see [Spall et al., 2006]. A more empirical way is the benchmarking of algorithms on testbeds of functions with known properties. Langdon and Poli [2007] has proposed an improvement on the benchmarking experimental

framework by using Genetic Programming to generate problems that would show the advantages and drawbacks of a few algorithms including CMA-ES, DE, PSO, and a Newton method.

The practitioner is provided with many different benchmarking systems for which we give some elements of comparison here. These benchmarks are usually addressed to a specific field of research, and provide tools that find their purpose at different levels from testbeds and experimental framework to application programming interfaces (APIs) which allow the BBO practitioner to make their own pieces of software compatible with the benchmarks.

2.2.1 Optimisers

We present here some optimisation software frameworks which are inscribed in a much broader scope than that of BBO. As a part of these software frameworks, some optimisers are proposed for solving BBO problems.

2.2.1.1 NLOPT

NLopt[3] [Johnson, 2008] provides with a unified software framework for some state-of-the-art derivative-free algorithms. It was built out of the necessity to test and compare algorithms. NLopt provides with open-source implementation of some BBO algorithms as well as some test functions.

2.2.1.2 COIN-OR

The Computational Infrastructure for Operations Research (COIN-OR)[4] project is an open-source software presented as a collection of projects related to optimisation [Lougee-Heimer, 2003]. As the project name states, the COIN-OR project is born and made for people from the field of Operational Research. The COIN-OR project provides with tools for building the piece of software needed to run the different projects, tools for outputting graphs, APIs. Development is still ongoing as the projects are at different levels of maturity.

[3] http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms
[4] http://www.coin-or.org

The DFO[5] and Ipopt[6] projects which are under the deterministic non-linear optimisation category are the most closely related to our interests. Both projects propose algorithm to solve BBO problems. The DFO project focuses on the implementation of the DFO algorithm [Conn et al., 1997a,b, 1998] which is a trust region method, see Section 2.1.1.2, adapted to BBO. The project Ipopt is still active and the underlying algorithm is a quasi-Newton method in the case of BBO [Wächter and Biegler, 2006]. None of the two projects provide with benchmarks. Only a few test functions are provided by DFO. Ipopt proposes interfaces to modelling languages such as AMPL[7] used to program optimisation problems or to testing environment such as CUTEr.

2.2.2 Testbeds

A number of projects proposes optimisation testbeds regrouping many test functions from real world applications or academic background.

2.2.2.1 CUTEr

The testing environment Constrained and Unconstrained Testing Environment, revisited (CUTEr)[8] [Gould et al., 2003] proposes some BBO test functions. The test problems of CUTEr are written using the SIF modelling language[9].

Current developments of CUTEr propose new interfaces including one improved interface to MATLAB. The CUTEr provides with BBO problems which can be of great use to practitioners, other BBO benchmarking frameworks have used problems from CUTEr [Moré and Wild, 2008]. Usually they are restricted to dozens of problems of CUTEr, which overall has more than a thousand of problems though many of them are from constrained optimisation which is not in the scope of BBO.

Though the problems in CUTEr are all listed on the web[10], details of an optimisation problem are only available by looking into the SIF file of the problem. Instead, a classification scheme[11] is used to provide with information on the problem: 1. the type

[5] https://projects.coin-or.org/Dfo
[6] https://projects.coin-or.org/Ipopt
[7] A Modeling Language for Mathematical Programming, http://www.ampl.com/
[8] http://hsl.rl.ac.uk/cuter-www/
[9] http://www.numerical.rl.ac.uk/lancelot/sif/sifhtml.html
[10] http://cuter.rl.ac.uk/cuter-www/Problems/mastsif.shtml
[11] http://cuter.rl.ac.uk/cuter-www/Problems/classification.shtml

of objective function, for instance constant or quadratic, 2. the type of constraints, 3. the smoothness of the problem which is determined by the fact that the function is twice differentiable and continuous, 4. the origin of the optimisation problem whether it is from academics or real world problems, 5. the dimension of the problem (which can be varied for some problems).

2.2.2.2 COCONUT

COntinuous CONstraint-Updating the Technology (COCONUT)[12] proposes to integrate techniques from mathematical programming, constraint programming and interval analysis into a single algorithm. COCONUT proposed a framework with a strategy engine at its core that calls modules, which organise models of the optimisation into a search graph and try to solve the sub-problems associated to the restrained models in the search graph.

The project comes with the COCONUT benchmark[13] which is a collection of problems for testing the algorithms in COCONUT. The benchmark includes some problems from CUTEr, see Section 2.2.2.1, as well as some from other sources from the field of global optimisation.

2.2.2.3 CEC 2005 Special Session

The community of evolutionary computation found in the special session on real-parameter optimization CEC 2005 [Suganthan et al., 2005], an experimental framework aiming for the rigourous and fair comparison of algorithms as well as an implementation that proposes twenty-five problems coded in C, MATLAB and JAVA. The results of a number of algorithms on the test bed of CEC 2005 are compiled in [Hansen, 2006b].

2.2.3 Benchmarking software

Overall, the practitioner in BBO is left to decide how the algorithms considered are to be compared. The performance comparison of algorithms depends on the criteria considered and it is very probable that these criteria are different for most people.

[12] http://www.mat.univie.ac.at/users/neum/public_html/glopt/coconut/
[13] http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html

In this context, some software exists that can help the practitioners implement their own benchmarking instance.

2.2.3.1 NEOS

NEOS[14] is a server for optimisation providing many pieces of optimiser software and an extensive list of optimisation problems including CUTEr, see Section 2.2.2.1. NEOS works using queries made to the server returning the results of optimisers on the chosen problem. NEOS is the joint effort of many researchers in the field of optimisation and operations research. NEOS provides distributed computational power for the public to test algorithms on test functions which are altogether hosted on the server. New algorithms and problems can be submitted.

2.2.3.2 Other projects

Many different projects exist and are in the developmental state. The Optimization Algorithm Toolkit (OAT)[15] and LOPTI[16] both are single-man attempts at the benchmarking problem we are considering. The OAT [Brownlee, 2007a,b] resembles very much to Weka[17] which provides with an interface and all the necessary software for practitioners in the field of machine learning to test different algorithms on a collection of real-world data mining problems. The OAT proposes the same experience in the field of combinatorial and continuous optimisation. LOPTI is in a developmental state and proposes a benchmarking framework of four optimisers facing BBO problems. The Java COntinuous Optimization Library (JCOOL)[18] also addresses the benchmarking problem, the prototype of the library has been used in the context of Group of Adaptive Model Evolution (GAME), from the same authors, which uses optimisation techniques to evolve machine learning models [Pavel Kordík, 2007].

[14] http://www-neos.mcs.anl.gov/
[15] http://optalgtoolkit.sourceforge.net
[16] http://volnitsky.com/project/lopti/
[17] http://sourceforge.net/projects/weka/
[18] http://cig.felk.cvut.cz/projects/jcool/

2.3 Discussion of the Review of the State of the Art

In this Chapter, we have presented a list of different methods from the fields of Operational Research, Global Optimisation and Evolutionary Computation. Theoretical approaches have been presented as well as experimental heuristics, the former delivering proofs of convergence under certain assumptions and the latter focusing on delivering suitable solutions in practice.

The many interactions between operational research and global optimisation are at the origin of the majority of the number of publications and the joint works presented Section 2.2. Many efforts still need to be made so that evolutionary computation have the same level of interaction with the first two fields.

Since there is a large quantity benchmarking software, the practitioner in BBO has many opportunities for testing either a prototypical algorithm or for trying different optimisers on an objective function. Results of the prototypical algorithm can be compared to those appearing in the abundant literature associated to the benchmarking software. Results of different algorithms on a test function may bring to light some characteristics of the problem considered.

We are only aware of few attempts at obtaining comparable baseline results of numerous algorithms —more than ten— on numerous problems —again more than ten. The need for the comparison results of so many algorithms can be discussed — the same way the need for the benchmarking architecture that would generate these comparison results can be discussed. Nevertheless, having some baseline comparisons could help the practitioner decide when facing the 'grey-box' or 'black-box' scenario. The results of benchmarking a new optimisation problem, in the case of the 'grey-box scenario' where some information is available, could provide with an overall idea which algorithms are appropriate for their problem. In the case of the 'black-box' scenario, how other algorithms perform on the test function considered might provide information on the problem.

Chapter 3

CMA-ES Variants

This chapter presents and extends the work published in [Ros and Hansen, 2008a,b].

Contents

3.1 Introduction

The search space dimensionality, n, plays an essential role in real parameter $\mathbb{R}^n$ optimisation, where a non-linear *objective function*, $f : \mathbb{R}^n \to \mathbb{R}$, is to be minimised. Its importance is emphasised by the notion of *curse of dimensionality*: the search space volume increases exponentially with n, making space filling sampling intractable even for moderate dimensionalities. The curse of dimensionality is only a concern if *dependencies* between parameters of the objective function are prevalent: when the parameters are independent, the search can be conducted along coordinate axes in *one-dimensional* subspaces altogether by n one-dimensional search procedures. Consequently, difficult real parameter optimisation problems exhibit essential dependencies between the parameters—and learning dependencies turns out to be decisive for solving these difficult problems. In evolutionary computation the issue of learning dependencies in real parameter search spaces is successfully addressed by covariance matrix adaptation (CMA) [Hansen and Ostermeier, 2001]. The CMA learns all pair-wise dependencies between all parameters by updating a covariance matrix for the sample distribution. The update mechanism is independent of the given coordinate system. The CMA was introduced for evolution strategies (ESs) but recently applied also in Evolutionary Gradient Search [Arnold and Salomon, 2007]. In learning *all* pair-wise dependencies, the CMA algorithm has an internal computational complexity of at least $\mathcal{O}(n^2)$. Empirical results indicate that, in order to learn the complete covariance matrix, the *number of objective function evaluations* usually scales sub-quadratically with n [Hansen and Ostermeier, 2001, Hansen et al., 2003].

As was already stated, we assume a black-box scenario in which evaluations of the function f are the only way to gather insights into the nature of f (and therefore to make a reasonable proposal for a solution vector with small function value). The number of function evaluations is regarded as *search costs*. Furthermore, we call a function f *separable* if the parameters of f are independent in that the global optimum can be obtained by n one-dimensional optimisation procedures along the coordinate axes for any given initial point.

3.1.1 Motivation of this Contribution

A principle limitation of CMA results from the number of so-called strategy parameters, $\frac{n^2+n}{2}$, that needs to be adapted in the covariance matrix. The number of strategy parameters, in other words the degrees of freedom in the covariance matrix, can dominate the search costs (number of objective function evaluations to reach a target function value). The full learning task scales roughly with n^2 (see *e.g.* [Hansen and Ostermeier, 2001]). Therefore, for large search space dimensionalities, achieving a better scaling property might be attractive. A second limitation of CMA, probably less important, lies in its *internal* computational complexity and is explained in the following.

- Sampling a general multivariate normally distributed random vector has a complexity of n^2 (per sampled n-dimensional vector). A matrix-vector multiplication needs to be conducted.

- Updating the covariance matrix has a complexity of $(\mu + 1)n^2$. The so-called rank-μ update [Hansen et al., 2003] amounts to μ covariance matrix updates, one for each parent vector.

- Factorising the covariance matrix $\boldsymbol{C}$ into $\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{C}$ has a complexity of n^3. The factorisation is needed to sample the multivariate normal distribution with covariance matrix $\boldsymbol{C}$. In the CMA-ES, usually an eigendecomposition is used to compute a *symmetric* (unique) factorisation matrix $\boldsymbol{A}$. A symmetric factorisation allows to compute the conjugate evolution path for step-size adaptation accurately, and it allows to track the eigenvalues of the covariance matrix, which often proves to be very useful in practice. Also the eigendecomposition has a complexity of n^3. Usually this computation is postponed until after $n/10$ generations and consequently slightly outdated distributions are sampled (with an insignificant effect on the performance) [Hansen and Ostermeier, 2001]. Consequently the complexity of this step becomes n^2 per generation.[1]

[1] More precisely, the computation is postponed until after ${c_{\text{cov}}}^{-1}n^{-1}/5$ generations, where the learning rate for the covariance matrix, c_{cov}, equals approximately $2n^{-2}$ for small populations. As the learning rate depends on the parent population size, the complexity becomes n^2 per parent vector.

In summary, several steps in the CMA algorithm have a computational complexity of $\Theta(n^2)$.

The most obvious option toward achieving a better scaling behaviour for the search costs is to *reduce the degrees of freedom* in the covariance matrix. We can think of several ways and parametrisation to reduce the degrees of freedom, resulting in a family of potentially useful modifications of CMA-ES, which *trade off model complexity for learning speed*. This trade will be a bad buy whenever the full model complexity is indispensable in order to efficiently solve the underlying problem. Otherwise, search costs can be reduced according to a reduced learning period. Therefore determining which correlations between variables of the optimisation problem considered is an interesting issue. *Linkage learning* and also to some extent sensitivity analysis or feature selection for instance, focuses on the determination of these essential correlations. The covariance matrix in the CMA-ES also provides us with another representation of the essential correlations of the problem.

In this chapter, we will assume that learning the full model can be superfluous and pursue modifications of CMA that reduces the degrees of freedom in the covariance matrix. These modifications denoted as sep-CMA and block-CMA enable to learn the block diagonal elements of the covariance matrix. Though these modifications can be interpreted as a preliminary step, it reveals some interesting perspectives on its own.

- The sep-CMA-ES can serve as a *baseline comparison*. Comparing with the CMA-ES, the profits and losses from learning *all the dependencies* in the CMA framework can be measured.

- On non-trivial *large scale problems* (say $n \approx 1000$ or larger) with only moderate dependencies, sep-CMA-ES and block-CMA-ES become valuable alternatives to CMA-ES, because they might perform considerably faster. On fast to evaluate objective functions, also a large advantage will be obtained when regarding the overall CPU-time. It is important to note that both variants sep-CMA-ES and block-CMA-ES will be able to exploit a *large population size* just like CMA-ES [Hansen and Kern, 2004, Hansen et al., 2003].

Objectives of this Chapter In this chapter we address the following objectives.

- Presenting the smallest possible modification of CMA that can learn a scaling of parameters in linear time, and as an important part of algorithm design, *carefully re-identifying the learning rate for the covariance matrix.*
- Extending this modification by considering a *block-diagonal CMA* instead of the diagonal CMA.
- Comparing the performance of these modified algorithms to CMA-ES on both separable and *non-separable* functions.

3.1.2 Favourably Scaling CMA Variants: Previous Works

The learning rate for the covariance matrix in CMA-ES is roughly proportional to n^{-2}, meaning the adaptation of the full covariance matrix needs roughly $\mathcal{O}(n^2)$ function evaluations. Nevertheless, a constant number of long axes of the distribution can be learnt in $\mathcal{O}(n)$, that is, in a linear number of function evaluations: the so-called *cumulation* allows for learning subspaces, in the space of covariance matrices, in linear time. Therefore the scale-up for the number of objective function evaluations is linear on the cigar function and on smooth ridge functions [Hansen and Ostermeier, 2001, Hansen et al., 2003].

Some previous approaches reduce the overall time complexity of CMA-ES.

Some ESs, which were introduced prior to CMA-ES, already implemented key features of the CMA-ES and scale favourably with the dimension. In [Ostermeier et al., 1994], a derandomised mutative step-size procedure was introduced to adapt individual step-sizes using cumulation and resulted in the indi-ES which is a $(1, \lambda)$-ES with n strategy parameters. An extension of this derandomised step-size adaptation denoted AII-ES and introduced in [Hansen et al., 1995] consists in combining it with the adaptation of one direction. This ES updates $2n$ strategy parameters.

The MVA-ES algorithm [Poland and Zell, 2001] uses the adaptation of the main (mutation) vector. This modification renders the time complexity of the algorithm to $\mathcal{O}(n)$ as the strategy parameters of the adaptation process are reduced to the length of the main vector, n. The MVA-ES algorithm is efficient in the specific case of objective functions having a single preferred mutation direction.

Filling the gap between CMA-ES and MVA-ES, L-CMA-ES [Knight and Lunacek, 2007] is a variant in which a parameter m allows to control the dimensionality of

the representation of the mutation distribution. The optimisation of the initial n-dimensional problem is restrained to that of its m main components. For the two extremes, if $m = 1$, L-CMA-ES is similar in substance to MVA-ES and if $m = n$, it is equivalent to the original CMA-ES.

A $\mathcal{O}(n^2)$ incremental Cholesky update of the covariance matrix was proposed as a replacement for the eigendecomposition in CMA [Igel et al., 2006] The use of the incremental Cholesky update was made compatible with the concept of evolution path [Suttorp et al., 2009] and has been extended to use larger population size. The resulting CMA-ES variant is a rank-one CMA-ES which benefits from the features of the CMA with a comparably better time complexity. Otherwise, with respect to numerical stability in the problems that we consider in this chapter, the use of eigen-decomposition or Cholesky decomposition poses no problem. Only with problems with a very high condition number (a notion introduced in Section 3.4) do we expect numerical errors being an experimental issue.

In this chapter, as opposed to previous works, we address another subspace of strategy parameters that can be easily identified: the diagonal blocks of the covariance matrix.

The remainder of this chapter is organised as follows. The Section 3.2 presents the CMA-ES algorithm and from this description, we introduce sep-CMA-ES and block-CMA-ES in Section 3.3. We explain how the problem of reducing the number of strategy parameters was addressed as well as compare the time complexity of sep-CMA-ES to that of CMA-ES. In Section 3.4, we propose to analyse sep-CMA-ES and block-CMA-ES on some standard test functions. Results from these experiments are discussed in Section 3.5 and provide insights that we develop in the last section of this chapter.

3.2 CMA-ES

The CMA-ES is a state-of-the-art continuous domain evolution strategy algorithm, introduced by [Hansen and Ostermeier, 2001] and described in [Hansen and Kern, 2004, Hansen et al., 2003] that uses a covariance matrix adaptation combined with the computation of an evolution path. The CMA-ES benefits from larger population sizes by the use of a rank-μ update along with a weighted recombination of offspring.

It is more precisely denoted as $(\mu/\mu_W, \lambda)$ CMA-ES. Default values of the algorithm parameters are discussed in detail in [Hansen and Ostermeier, 2001].

Offspring for the generation $g+1$ are sampled according to the following equation:

$$\boldsymbol{x}_k^{(g+1)} \sim \langle \boldsymbol{x} \rangle_{\mathrm{W}}^{(g)} + \sigma^{(g)} \underbrace{\boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} \boldsymbol{z}_k^{(g+1)}}_{\mathcal{N}\left(0, \boldsymbol{C}^{(g)}\right)}, \; k = 1, \ldots, \lambda \tag{3.1}$$

where:

- λ is the population size, its default value being $4 + \lfloor 3 \ln(n) \rfloor$;
- μ is the number of the best individuals that is recombined, the default value of μ is $\lfloor \frac{\lambda}{2} \rfloor$;
- $\langle \boldsymbol{x} \rangle_{\mathrm{W}}^{(g)} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(g)}$ is the weighted mean of the μ best individuals at generation g, $\boldsymbol{x}_{i:\lambda}^{(g)}$ denotes the i-th best out of the λ individuals ranked by function value;
- (w_i), $i = 1, \ldots, \mu$ are the recombination weights, they are positive and sum to one. Setting the w_i to $1/\mu$ corresponds to an intermediate recombination. We use this expression: $w_i = \frac{\ln(\mu+1) - \ln(i)}{\sum_{j=1}^{\mu} \ln(\mu+1) - \ln(j)}$ that favours the best ranked individuals more;
- $\mathcal{N}(0, \boldsymbol{M})$ denotes independent realisations of the multi-variate normal distribution with covariance matrix $\boldsymbol{M}$;
- the random vectors $\boldsymbol{z}_k^{(g+1)}$ are $\mathcal{N}(0, \boldsymbol{I})$ distributed, and just as for the $\boldsymbol{x}_k^{(g)}$, we can compute their weighted mean: $\langle \boldsymbol{z} \rangle_{\mathrm{W}}^{(g+1)} = \sum_{i=1}^{\mu} w_i \boldsymbol{z}_{i:\lambda}^{(g+1)}$, $\boldsymbol{z}_{i:\lambda}^{(g+1)}$ denotes the i-th best out of the λ individuals ranked by function value;
- $\boldsymbol{B}^{(g)}$ is an orthogonal $n \times n$ matrix and $\boldsymbol{D}^{(g)}$ is a diagonal $n \times n$ matrix obtained from the eigendecomposition of $\boldsymbol{C}^{(g)}$, $\boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} \left(\boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)}\right)^T = \boldsymbol{C}^{(g)}$. The covariance matrix, $\boldsymbol{C}^{(g)}$, is symmetric positive definite, its default initial value is $\boldsymbol{I}$;
- $\sigma^{(g)} \in \mathbb{R}^+$ is the step-size.

The parameter $\langle \boldsymbol{x} \rangle_{\mathrm{W}}^{(0)}$ is problem-dependent.

Both the global step-size $\sigma^{(g)}$ and the covariance matrix $\boldsymbol{C}^{(g)}$ are iteratively adapted. The path length control consists in adapting the global step size $\sigma^{(g)}$ by an evolution path.

$$\boldsymbol{p_\sigma}^{(g+1)} = (1-c_\sigma)\boldsymbol{p_\sigma}^{(g)} + \sqrt{c_\sigma(2-c_\sigma)} \underbrace{\sqrt{\mu_{\text{eff}}}\boldsymbol{B}^{(g)}\langle \boldsymbol{z}\rangle_{\text{W}}^{(g+1)}}_{\frac{\sqrt{\mu_{\text{eff}}}}{\sigma^{(g)}}\boldsymbol{C}^{(g)^{-\frac{1}{2}}}\left(\langle \boldsymbol{x}\rangle_{\text{W}}^{(g+1)}-\langle \boldsymbol{x}\rangle_{\text{W}}^{(g)}\right)} \tag{3.2}$$

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\parallel \boldsymbol{p_\sigma}^{(g+1)} \parallel}{E(\parallel \mathcal{N}(0,\boldsymbol{I}) \parallel)} - 1\right)\right) \tag{3.3}$$

where:

- $c_\sigma \in \,]0,1]$ is the time constant for the adaptation of the step size $\sigma^{(g+1)}$, its default value is: $\frac{\mu_{\text{eff}}+2}{n+\mu_{\text{eff}}+3}$;
- $\mu_{\text{eff}} = (\sum_{i=1}^{\mu} w_i)^2 / \sum_{i=1}^{\mu} w_i^2$ denotes the 'variance-effective selection mass', μ_{eff} is equal to μ if $w_i = 1/\mu$;
- $d_\sigma > 0$ is a damping factor, its default value is: $d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{\text{eff}}-1}{n+1}} - 1\right) + c_\sigma$;

The initial value of the evolution path is $\boldsymbol{p_\sigma}^{(0)} = \boldsymbol{0}$. The initial step-size $\sigma^{(0)}$ is a problem-dependent parameter.

The adaptation of $\boldsymbol{C}^{(g)}$ is done by the evolution path $\boldsymbol{p}_{\text{c}}{}^{(g+1)}$ and by the μ-weighted difference vectors between the recent parents and $\langle \boldsymbol{x}\rangle_{\text{W}}^{(g)}$:

$$\boldsymbol{p}_{\text{c}}{}^{(g+1)} = (1-c_{\text{c}})\boldsymbol{p}_{\text{c}}{}^{(g)} + H_\sigma^{(g+1)}\sqrt{c_{\text{c}}(2-c_{\text{c}})} \underbrace{\sqrt{\mu_{\text{eff}}}\boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)}\langle \boldsymbol{z}\rangle_{\text{W}}^{(g+1)}}_{\frac{\sqrt{\mu_{\text{eff}}}}{\sigma^{(g)}}\left(\langle \boldsymbol{x}\rangle_{\text{W}}^{(g+1)}-\langle \boldsymbol{x}\rangle_{\text{W}}^{(g)}\right)} \tag{3.4}$$

$$\begin{aligned}\boldsymbol{C}^{(g+1)} = (1-c_{\text{cov}})\boldsymbol{C}^{(g)} &+ \frac{1}{\mu_{\text{cov}}} c_{\text{cov}} \boldsymbol{p}_{\text{c}}{}^{(g+1)}\left(\boldsymbol{p}_{\text{c}}{}^{(g+1)}\right)^T \\ &+ c_{\text{cov}}\left(1-\frac{1}{\mu_{\text{cov}}}\right) \underbrace{\sum_{i=1}^{\mu} w_i \boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)}\boldsymbol{z}_{i:\lambda}^{(g+1)}\left(\boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)}\boldsymbol{z}_{i:\lambda}^{(g+1)}\right)^T}_{\sum_{i=1}^{\mu}\frac{w_i}{\sigma^{(g)^2}}\left(\boldsymbol{x}_{i:\lambda}^{(g+1)}-\langle \boldsymbol{x}\rangle_{\text{W}}^{(g)}\right)\left(\boldsymbol{x}_{i:\lambda}^{(g+1)}-\langle \boldsymbol{x}\rangle_{\text{W}}^{(g)}\right)^T}\end{aligned} \tag{3.5}$$

where:

- $c_c \in [0, 1]$ has the same role as c_σ in Equation (3.2), it is a time a constant for the adaptation of the covariance matrix, its default value is $\frac{4}{n+4}$;
- $H_\sigma^{(g+1)}$ equals to one if $\frac{\|\boldsymbol{p_\sigma}^{(g+1)}\|}{\sqrt{1-(1-c_\sigma)^{2(g+1)}}} < (1.4 + \frac{2}{n+1})E(\| \mathcal{N}(0, \boldsymbol{I}) \|)$, and zero otherwise. The condition on H_σ is slightly different from [Hansen and Kern, 2004], which can be noticeable in case of large dimension and small λ, say 9, together with either a too small initial step-size σ or on time-variant objective functions.
- $\frac{1}{\mu_{\text{cov}}} \in [0, 1]$ is a coefficient that controls the emphasis on the evolution path term, μ_{cov} default value is μ_{eff};
- $c_{\text{cov}} \in [0, 1]$ is the learning rate, its default value[2] is:

$$c_{\text{cov default}} = \frac{1}{\mu_{\text{cov}}} \frac{2}{\left(n + \sqrt{2}\right)^2} + \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \min\left(1, \frac{2\mu_{\text{cov}} - 1}{(n+2)^2 + \mu_{\text{cov}}}\right) \tag{3.6}$$

The initial values are: $\boldsymbol{p}_c^{(0)} = \boldsymbol{0}$, $\boldsymbol{C}^{(0)} = \boldsymbol{I}$.

The whole process of sampling new individuals (Equation (3.1)) and updating the internal strategy parameters (covariance matrix) (Equations (3.2) to (3.5)) is iterated until a stopping criterion is reached.

3.3 CMA-ES Variants with Reduced Time and Space Complexity

In this section we introduce the two variants of the CMA-ES, sep-CMA-ES and block-CMA-ES, designed to reduce the time complexity of the adaptation of the covariance matrix $\boldsymbol{C}^{(g)}$.

3.3.1 Description of sep-CMA-ES

Compared to the default CMA-ES, the covariance matrix $\boldsymbol{C}$ is constrained to be diagonal. This results in having an ES that samples independently w.r.t. the coordinate system using n individual variances that are updated.

[2] The Equation (3.6) differs from [Hansen and Kern, 2004].

The update of the covariance matrix in Equation (3.5) is modified: the covariance matrix $\boldsymbol{C}^{(g)}$ remains a diagonal matrix, because its update is restrained to the diagonal elements of the matrix, and Equation (3.5) becomes:

$$c_{jj}^{(g+1)} = (1 - c_{\text{cov}})c_{jj}^{(g)} + \frac{1}{\mu_{\text{cov}}} c_{\text{cov}} \left(p_c^{(g+1)}\right)_j^2 + c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i \, c_{jj}^{(g)} \left(z_{i:\lambda}^{(g+1)}\right)_j^2, \quad j = 1, \ldots, n \quad (3.5')$$

where, for $j = 1, \ldots, n$, the c_{jj} are the diagonal elements of $\boldsymbol{C}^{(g)}$ and the $\left(z_{i:\lambda}^{(g+1)}\right)_j$ are the j-th component of $\boldsymbol{z}_{i:\lambda}^{(g+1)}$.

Whereas in the CMA-ES an eigendecomposition was needed before Equation (3.1), now this step is reduced to taking the square root of the diagonal elements of $\boldsymbol{C}^{(g)}$, which has $\mathcal{O}(n)$ time complexity. The complexity of all other equations involving matrix operations is reduced since they now involve vector operations. The complexity reduction is made possible by the reduction of the model complexity but in the process it loses what made CMA-ES able to learn the dependencies of the parameters of the objective function. In other words, sep-CMA-ES loses the property of being rotationally invariant that CMA-ES has.

The study presented in [Ros and Hansen, 2008a,b] provided with an identification of an increased learning rate by a factor of $\frac{n+2}{3}$ for which experimental results were provided.

3.3.2 Description of block-CMA-ES

In this section we introduce another CMA-ES variant, denoted as block-CMA-ES, also designed to reduce the time complexity of the adaptation of the covariance matrix $\boldsymbol{C}^{(g)}$. Compared to the sep-CMA-ES and CMA-ES, the covariance matrix $\boldsymbol{C}$ is constrained to be block diagonal.

For obtaining the $[m_1, m_2, \ldots]$-block-CMA-ES from the CMA-ES algorithm, the Equation (3.5) for the update of the covariance matrix is applied to each block j:

$$\boldsymbol{C}_j^{(g+1)} = (1 - c_{\text{cov}})\boldsymbol{C}_j{}^{(g)} + \frac{1}{\mu_{\text{cov}}} c_{\text{cov}} \left(\boldsymbol{p}_{\text{c}}{}^{(g+1)}\right)_j \left(\boldsymbol{p}_{\text{c}}{}^{(g+1)}\right)_j^T + c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i \boldsymbol{B}_j^{(g)} \boldsymbol{D}_j^{(g)} \left(\boldsymbol{z}_{i:\lambda}^{(g+1)}\right)_j \left(\boldsymbol{B}_j^{(g)} \boldsymbol{D}_j^{(g)} \left(\boldsymbol{z}_{i:\lambda}^{(g+1)}\right)_j\right)^T \quad (3.5")$$

where:

- the covariance matrix $\boldsymbol{C}^{(g)}$ is block diagonal

$$\begin{pmatrix} \boldsymbol{C}_1^{(g)} & & 0 \\ & \boldsymbol{C}_2^{(g)} & \\ 0 & & \ddots \end{pmatrix}$$

  with the $\boldsymbol{C}_j^{(g)}$ being matrices of size $m_j \times m_j$,

- $\left(\boldsymbol{p}_{\text{c}}{}^{(g+1)}\right)_j$, $\boldsymbol{B}_j^{(g)}$, $\boldsymbol{D}_j^{(g)}$, $\left(z_{i:\lambda}^{(g+1)}\right)_j$ are the respective projections of $\boldsymbol{p}_{\text{c}}{}^{(g+1)}$, $\boldsymbol{B}^{(g)}$, $\boldsymbol{D}^{(g)}$, $z_{i:\lambda}^{(g+1)}$ onto the subspace considered.

The covariance matrix $\boldsymbol{C}^{(g)}$ remains a block diagonal matrix, because its update is restrained to the block diagonal elements of the matrix. We will from this point on use the bracketed notation $[m_1, m_2 \ldots]$ to denote block configurations of block-CMA-ES.

The model complexity can be reduced depending on the number and the size of the blocks. Using the full covariance matrix with a learning rate equal to $c_{\text{covdefault}}$ the $[n]$-block-CMA-ES algorithm is identical to CMA-ES. Using only blocks of size one and a learning rate of $\frac{n+2}{3} c_{\text{covdefault}}$ the $[1, \ldots, 1]$-block-CMA-ES is the same as the sep-CMA-ES algorithm [Ros and Hansen, 2008a,b]. Whereas in the CMA-ES the eigendecomposition of an $n \times n$ matrix was needed before Equation (3.1), now the complexity of this step can be diminished depending on the sizes of the blocks to a $\mathcal{O}(n)$ time complexity at best. The complexity of all other steps in the block-CMA-ES algorithm can be decreased accordingly.

3.3.3 Identification of $c_{\mathbf{cov}}$

According to [Hansen, 2009d], c_{cov} can be written as follows

$$c_{\text{cov}} = c_1 + c_\mu \tag{3.7}$$

$$c_1 = \frac{1}{\text{dof} + 2\sqrt{\text{dof}} + \frac{\mu_{\text{eff}}}{n}} \tag{3.8}$$

$$c_\mu = \min\left(1 - c_1, \frac{\alpha_\mu^0 + \mu_{\text{eff}} - 2 + \frac{1}{\mu_{\text{eff}}}}{\text{dof} + 4\sqrt{\text{dof}} + \frac{\mu_{\text{eff}}}{2}}\right) \tag{3.9}$$

$$\alpha_\mu^0 = 0.3 \tag{3.10}$$

where dof is the number of degrees of freedom in the covariance matrix. From our definition of sep-CMA-ES and block-CMA-ES, there are less degrees of freedom in the covariance matrix, $\text{dof} = \sum_j \frac{{m_j}^2 + m_j}{2}$ which ranges from n to $n + \frac{n^2 - n}{2}$. According to the Equations (3.7) to (3.10), dof being smaller implies the learning rate c_{cov} used in Equations (3.5') and (3.5") must be larger. We want to approximate this increase of the learning rate.

The behaviours of block-CMA-ES and by extension sep-CMA-ES are studied using different values for the learning rate. We use two measures to qualify the effects of the varying learning rate on block-CMA-ES: 1. the ratio of the square root of the largest and smallest eigenvalue of the final covariance matrix $\boldsymbol{C}^{(g)}$ (denoted in the following as *final axis ratio*) which should be close to the square root of the condition number of the objective function, 2. the number of function evaluations to reach a given target function value.

The uni-modal and separable sphere function, $f_{\text{sphere}}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$ was used as test function. We also tested the algorithm on the ellipsoid function,

$$f_{\text{elli}}^{\beta}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta^{\frac{i-1}{n-1}} y_i^2$$

where the parameter β is the condition number (ratio of the longest and smallest axis lengths) and $\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{x}$. The coordinate system $\boldsymbol{Q}$ is either equal to $\boldsymbol{I}$ for the axis-parallel function or, for the rotated function, to an orthogonal $n \times n$ matrix with each column vector $\boldsymbol{q}_j$ $(j = 1, \ldots, n)$ being a uniformly distributed unit vector.

The ellipsoid function is non-separable in the general case, convex quadratic and uni-modal.

3.3.3.1 Experimental Set-up for the Identification of $c_{\mathbf{cov}}$

The implementation of the block-CMA-ES algorithm that we use in our experiments is derived from a CMA-ES implementation in SCILAB[3]. We multiply the default value of the learning rate $c_{\text{covdefault}}$ (defined in Section 3.2) for an n-dimensional problem by a factor ranging from 0 to 160. The block-CMA-ES algorithm is tested in the $[1, \ldots, 1]$ variant and compared to the full model. We also test block-CMA-ES with the $[1, n-1]$ block configuration and see effect of varying the learning rate on unbalanced block configurations. Finally, same sized block configurations $[2, \ldots, 2]$, $[n/4, \ldots, n/4]$ and $[n/2, n/2]$ complete this investigation. Other than c_{cov}, the parameters of the algorithms are set to the default values given in Section 3.2 for both the CMA-ES and block-CMA-ES.

While the default initial value of the covariance matrix is $\boldsymbol{C}^{(0)} = \boldsymbol{I}$, CMA-ES and block-CMA-ES are also tested on the sphere function, f_{sphere}, with $\boldsymbol{C}^{(0)}$ set to a $n \times n$ diagonal matrix which diagonal elements are $(1, \beta^{\frac{1}{n-1}}, \beta^{\frac{2}{n-1}}, \ldots, \beta)$, with $\beta = 10^6$.

The dimension n of the problem is chosen in the interval $[2, 40]$. The initial distribution is centred on $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)}$ chosen uniformly in $[-20, 80]^n$ and the initial step-size is $\sigma^{(0)} = 100/3$. Runs fail if, before the target function value is attained, either the axis ratio is larger than 10^{16} or the maximum number of evaluations is reached. For each experimental set-up, eleven runs are done except for $c_{\text{cov}} = 0$ (no learning occurs) for which five runs are done. As for the coordinate system, $\boldsymbol{Q}$, we use eleven different rotation matrices.

3.3.3.2 Discussion on the Learning Rate in sep-CMA-ES and block-CMA-ES

Results are presented in Figures 3.1 to 3.4. For the larger values of c_{cov}, block-CMA-ES becomes less reliable since no run succeeds: they are stopped to prevent having precision issues in the eigendecomposition of covariance matrices with an axis ratio as large as 10^{16}. Results in Figure 3.1 on the sphere function show the final axis

[3] Available here: http://www.bionik.tu-berlin.de/user/niko/cmaesintro.html

ratios raise while the performance degrades (*i.e.* more function evaluations) when the learning rate increases.

On the sphere function, when the condition number of the covariance matrix is initialised to β instead of one, increasing the learning rate improves the performance until it degrades because the learning rate is too high.

Choosing the right learning rate means to make sure that the adaptation process is successful and that the performance of the algorithm is reliable. This is observable for intermediate values of c_{cov} where the final axis ratio is close to the condition number of the objective function which is one in the case of the sphere function. In the case of the ill-conditioned, non-separable ellipsoid function, $f_{\text{elli}}^{10^3}$, an intermediate learning rate returns final axis ratios that are comparable when the dimension varies and reliable performances as well.

Results of the CMA-ES for which the default value $c_{\text{covdefault}}$ is adjusted show the boundary values of the ratio $\frac{c_{\text{cov}}}{c_{\text{covdefault}}}$ to be pretty close whatever the dimension considered (top sub-figures in Figure 3.1, 3.2 and 3.3). Out of this domain, unreliable behaviours are to be expected. To have a similar domain as the dimension varies for the block-CMA-ES algorithm, the ratio $\frac{c_{\text{cov}}}{c_{\text{covdefault}}}$ had to be divided by $\frac{n+3/2}{\max_j(m_j)+2}$. We can see this normalisation make the results for the different block configurations very comparable, whether in terms of final axis ratio (top) or numbers of function evaluations (bottom sub-figures). The choice of the normalisation is conservative in the sense that it leaves for every dimension tested a margin of at least a factor of three before the learning rate c_{cov} is set too large. In all the following experiments, it is this default value, $c_{\text{cov}} = \frac{n+3/2}{\max_j(m_j)+2} c_{\text{covdefault}}$, that is used for sep-CMA-ES[4] and block-CMA-ES.

3.4 Test Functions and Methods

We describe the functions and methods used to test and compare sep-CMA-ES, block-CMA-ES and CMA-ES.

[4]This default value is smaller than the one defined in [Ros and Hansen, 2008a,b].

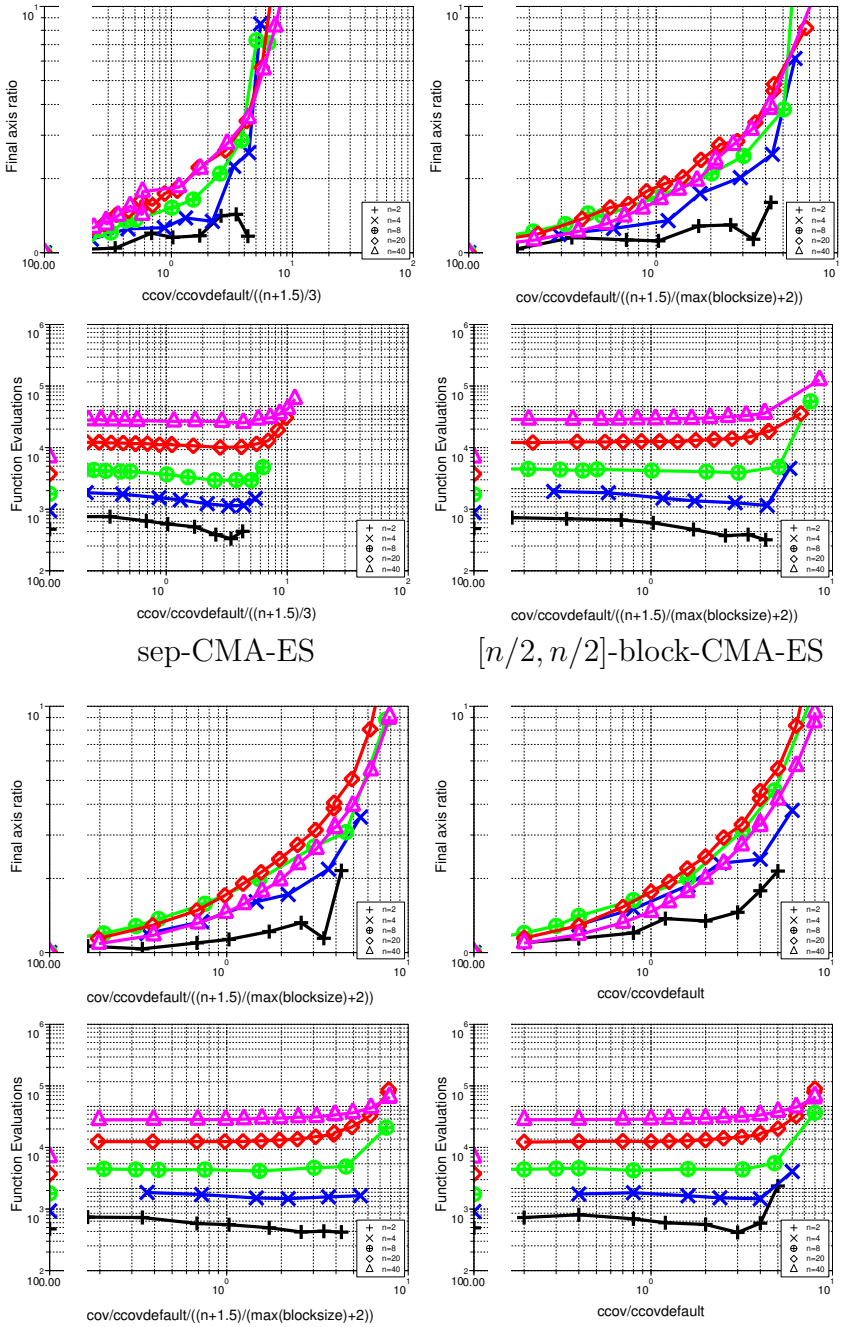
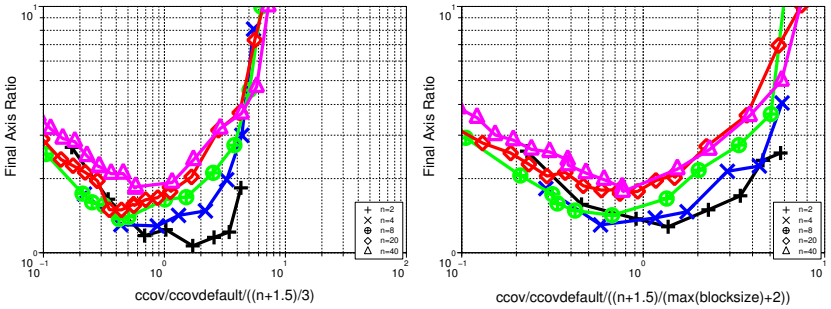
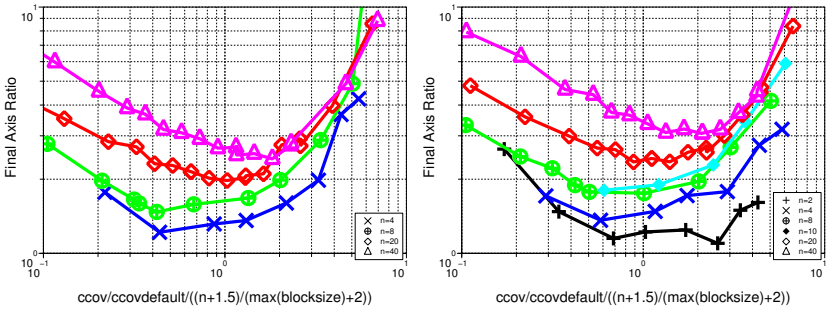


Figure 3.1: Results on the sphere function of variants of the CMA-ES with initial covariance matrix $\boldsymbol{C}^{(0)}$ set to $\boldsymbol{I}$ for problem dimension going from 2 to 40-D. A point is shown if, out of the 11 runs done for each set-up, all runs succeed in reaching the target function value of 10^{-9}. The top sub-figures show the median axis ratio of the final covariance matrix whereas the bottom sub-figures show the mean number of function evaluations.

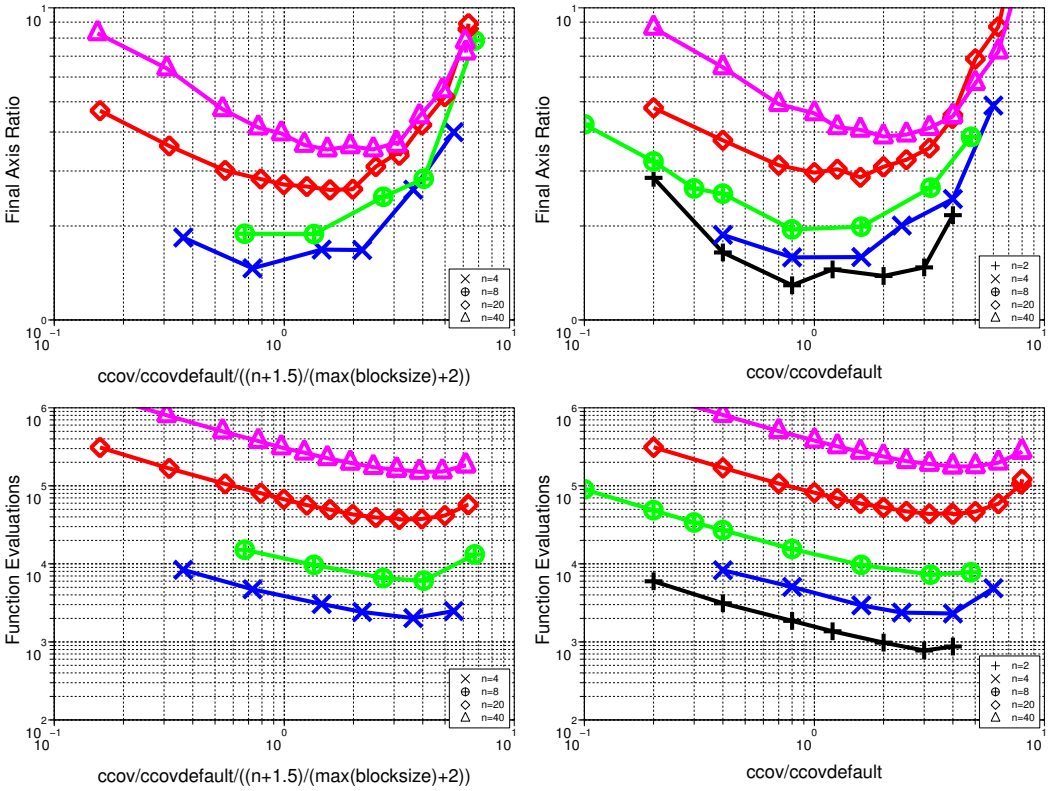


sep-CMA-ES $[2, \ldots, 2]$-block-CMA-ES



$[n/4, \ldots, n/4]$-block-CMA-ES $[n/2, n/2]$-block-CMA-ES

Figure 3.2: Results on the sphere function of variants of CMA-ES with an initial covariance matrix $\boldsymbol{C}^{(0)}$ being a diagonal matrix with diagonal elements $(10^6 \frac{i-1}{n-1})_{i=1,\ldots,n}$ for problem dimensions going from 2 to 40-D. A point is shown if, out of the eleven runs done for each set-up, all runs succeed in reaching the target function value of 10^{-9}. The top sub-figures show the median axis ratio of the final covariance matrix whereas the bottom sub-figures show the mean number of function evaluations. No success is observed for $c_{\text{cov}} = 0$.



$[1, n-1]$-block-CMA-ES CMA-ES

Figure 3.3: Results on the sphere function of variants of CMA-ES with an initial covariance matrix $\boldsymbol{C}^{(0)}$) being a diagonal matrix with diagonal elements $(10^{6\frac{i-1}{n-1}})_{i=1,\ldots,n}$ for problem dimensions going from 2 to 40-D. A point is shown if, out of the 11 runs done for each set-up, all runs succeed in reaching the target function value of 10^{-9}. The top sub-figures show the median axis ratio of the final covariance matrix whereas the bottom sub-figures show the mean number of function evaluations. No success is observed for $c_{\text{cov}} = 0$.

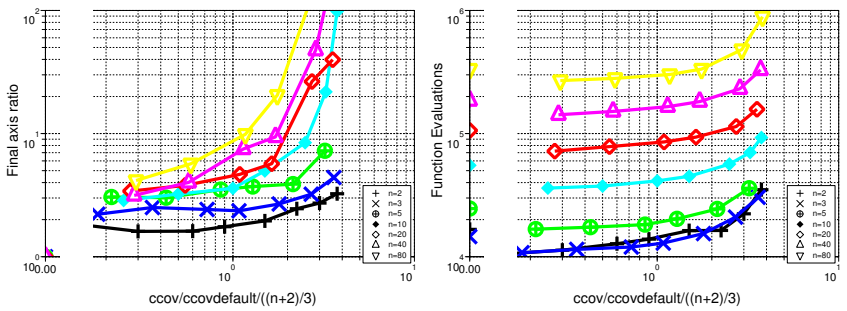


Figure 3.4: Results of sep-CMA-ES on the ellipsoid function ($\beta = 10^3$). A point is shown if, out of the 11 runs done for each set-up, all runs succeed in reaching the target function value of 10^{-9}. The sub-figure on the left shows the median axis ratio of the final covariance matrix whereas the sub-figure on the right shows the mean number of function evaluations.

3.4.1 Test Functions

All the functions are described in Table 3.1.

We have already introduced the Ellipsoid function in Section 3.3.3. The Cigar, Tablet, Two-Axes and Sum of Different Power functions (diffpow), denoted as f_{cigar}, f_{tablet}, f_{twoaxes} and f_{diffpow}, are all uni-modal and ill-conditioned. The conditioning of all of these functions can be controlled by a parameter β introduced in the Table 3.1. The parameter β works in the same way than that of the ellipsoid function with the notable exception of f_{diffpow} for which β is not the conditioning number of the function but merely a parameter to control the conditioning. These functions in their axis-parallel version are separable.

We also have the Rosenbrock function, f_{Rosen}, which is non-convex, not uni-modal for dimensions larger than 4-D and non separable. Again, a parameter β can be introduced to control the conditioning of the function.

All the previous functions are tested using their rotated version as well.

A variant of the ellipsoid function is tested: $f_{\text{hyperelli}}$ is used in [Ostermeier et al., 1994] to evaluate the performance of indi-ES. The function $f_{\text{hyperelli}}$ differs from the ellipsoid function by the fact that the condition number $\beta = n$ is fixed and is small compared to the condition numbers we consider for the previous functions. We use the function $f_{\text{hyperelli}}$ to compare our variants of CMA-ES to the indi-ES.

Table 3.1: Test functions. For all experiments on all functions, the initialisation range is $[-20, 80]^n$ and initial step-size is $\sigma^{(0)} = \frac{100}{3}$. For all functions, ellipsoid function, $\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{x}$, where $\boldsymbol{Q}$ is either $\boldsymbol{I}$ or an orthogonal $n \times n$ matrix with each column vector $\boldsymbol{q}_i$ being a uniformly distributed unit vector implementing an angle-preserving transformation.

Name	Function	f_{target}
Ellipsoid	$f^{\beta}_{\text{elli}}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta^{\frac{i-1}{n-1}} y_i^2$	10^{-9}
Rosenbrock	$f^{\beta}_{\text{Rosen}}(\boldsymbol{x}) = \sum_{i=1}^{n-1} \beta\,(y_i^2 - y_{i+1})^2 + (y_i - 1)^2$	10^{-9}
Diffpow	$f^{\beta}_{\text{diffpow}}(\boldsymbol{x}) = \sum_{i=1}^{n} \lvert y_i \rvert^{2+\beta\frac{i-1}{n-1}}$	10^{-14}
Cigar	$f^{\beta}_{\text{cigar}}(\boldsymbol{x}) = y_1^2 + \sum_{i=2}^{n} \beta y_i^2$	10^{-9}
Tablet	$f^{\beta}_{\text{tablet}}(\boldsymbol{x}) = \beta y_1^2 + \sum_{i=1}^{n} y_i^2$	10^{-9}
Two-axes	$f^{\beta}_{\text{twoaxes}}(\boldsymbol{x}) = \sum_{i=1}^{n/2} y_i^2 + \sum_{i=n/2+1}^{n} \beta y_i^2$	10^{-9}
Hyper-ellipsoid	$f_{\text{hyperelli}}(\boldsymbol{x}) = \sum_{i=1}^{n} (i\ y_i)^2$	10^{-10}

3.4.2 Experimental Set-up

All the algorithm parameters of the sep-CMA-ES and block-CMA-ES are set to their default value except for the learning rate.

CPU-Time Experiments The amount of CPU-time for the algorithms to reach a number of function evaluations is measured while the dimensionality of the problem varies: n ranges from 10 to 5120. We have implemented the sep-CMA-ES algorithm from the `purecmaes.m` MATLAB code[5].

We compare the sep-CMA-ES with variants of the CMA-ES algorithm that postpone the eigendecomposition until after $(c_{\text{cov}} n)^{-1} \alpha$ generations, $\alpha \in \{0, 0.1, 1\}$. These variants have also been implemented from the code in `purecmaes.m`. We make sure the number of function evaluations is large enough so that the eigendecomposition is computed at least ten times: 5×10^4 function evaluations when α is equal to 0.1 or 1 and the dimension is larger than 320, 10^4 otherwise. Three trials are done for each algorithm on each dimensionality. We test using two population sizes: $\lambda = 4 + \lfloor 3 \ln n \rfloor$ and $\lambda = 2n$. Experiments were performed on a single (no hyper-threading) Intel Core

[5] http://www.bionik.tu-berlin.de/user/niko/purecmaes.m

2 processor 2.66GHz with 2GB RAM.

Performance Experiments We use the SCILAB version of the block-CMA-ES algorithms. The test functions considered can be varied along two parameters: the dimension n and the condition number or parameter in the case of functions such as f_{diffpow} and f_{Rosen}. When n varies from two to a thousand and twenty-four, the condition number is set to 10^6, except for f_{diffpow} and f_{Rosen} for which the condition parameter is set to ten and a hundred respectively. When the condition number/parameter varies, the dimension of the search space is set to 16-D. The condition number then ranges from one to 10^{10}, the condition parameter of f_{diffpow} ranges between zero and ten, that of f_{Rosen} between one and 10^8.

The block-CMA-ES algorithm is tested in some of the regular block configurations used in Section 3.3.3. Those can be divided in three: the block configurations with fixed block sizes as the dimension increases ($[1, \ldots, 1]$ which is equivalent to sep-CMA-ES, $[2, \ldots, 2]$), those with a fixed number of blocks as the dimension increases ($[n/2, \ldots, n/2]$), $[n]$) and a block configuration which verifies none of the these two properties ($[\sqrt{n}, \ldots, \sqrt{n}]$). The CMA-ES is assimilated to the $[n]$-block-CMA-ES though the learning rates of the two are different.

For all problems, the starting point $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)}$ is chosen in $[-20, 80]^n$ and the initial step-size $\sigma^{(0)}$ is one third of the interval width. If the target function value given in Table 3.1 is reached within 10^7 function evaluations, a run is considered successful. Performances are averaged over eleven runs for the lower dimensions ($n < 100$), three runs for larger dimensions. The rotation matrix $\boldsymbol{Q}$ is changed for every single run.

In addition to this, we also examine previously published results: the ES algorithm with derandomised mutative step-size control [Ostermeier et al., 1994] denoted as indi-ES in the following, the AII-ES [Hansen et al., 1995], the MVA-ES [Poland and Zell, 2001] and the L-CMA-ES [Knight and Lunacek, 2007]. We use the same starting point, initial step-size (where applicable) and population sizes as those described in each of these cited works to compare with the results of sep-CMA-ES.

3.5 Results and Discussion

The dynamic behaviour of the block-CMA-ES algorithm is studied first. In Figure 3.5, the function value decreases steeply in the process of optimisation, even steeper after four thousand function evaluations when the adaptation of the larger eigenvalues has been done (shown in the bottom-left sub-figure). This adaptation of the eigenvalues corresponds to a *learning phase*, which in this case roughly represents half of the total search costs. Other observations are that: 1. the step-size decreases as expected (top-left sub-figure), 2. the diagonal elements of the covariance matrix all go to zero as well (bottom-right), 3. the axis ratio behaves correctly since, after four thousand function evaluations, the ratio of the largest and smallest axis lengths of $\boldsymbol{D}^{(g)}$ (bottom-left) is close to 10^3 which is the square root of β.

3.5.1 CPU-Time Experiments

The CPU-time per function evaluations versus the dimension for sep-CMA-ES and CMA-ES is displayed in Figure 3.6. For the default population size (top sub-figure), the time complexity of sep-CMA-ES scales like $n^{1.2}$ in the larger dimensions. In this context, if the eigendecomposition in CMA-ES is done at each iteration ($\alpha = 0$), the time complexity scales like $n^{2.7}$. The use of outdated covariance matrices reduces the time complexity to $n^{1.9}$, $n^{1.8}$ for $\alpha = 0.1, 1$ respectively, but sep-CMA-ES still outperforms CMA-ES by a factor of at least six in 100-D.

Experiments for $\lambda = 2n$ show that sep-CMA-ES achieves a linear time complexity for the larger dimensions. Again it clearly outperforms CMA-ES, this time by a factor of 10 when $n = 100$ and $\alpha = 1$ for which the time complexity scales with $n^{1.8}$ for dimensions up to 1000-D.

We would like to point out that the sep-CMA-ES, meaning $[1, \ldots, 1]$-block-CMA-ES, is especially advantaged in terms of time performance in MATLAB or SCILAB compared to block-CMA-ES using other block configurations. Indeed, due to the implementation of these programming language, processing vectors can be dramatically more efficient than looping over elements[6]. Therefore, in the case of the sep-CMA-ES dealing with the *vector* of the elements of the diagonal of the covariance matrix is more efficient than looping over the blocks of the block-CMA-ES in any other block

[6]See http://www.mathworks.com/support/tech-notes/1100/1109.html

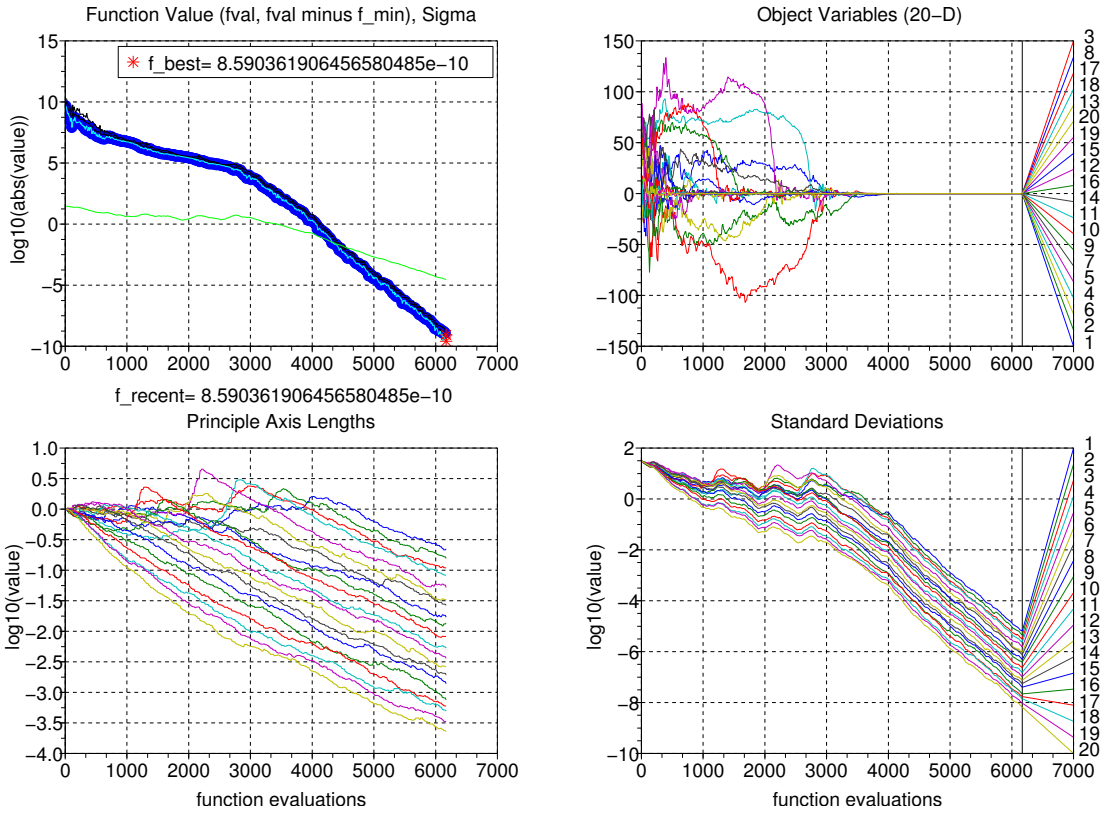


Figure 3.5: Single run of sep-CMA-ES on the axis-parallel ellipsoid function, $\beta = 10^6$, $n = 20$. The abscissa of all sub-figures is the time. Top-left is the evolution of the function values and of the step-size σ. Bottom-left is the evolution of the square root of the eigenvalues of the covariance matrix. Top-right is the evolution of the coordinates of the weighted mean of the individuals, bottom-right the evolution of the coordinate-wise standard deviations of the distribution of the population.

configuration.

3.5.2 Performance Experiments

On Separable Functions We are considering the dashed lines presented in Figures 3.7, 3.8 and 3.9 which represent the results of the different variants of the CMA-ES facing the axis-parallel version of the objective functions considered. The fact that the objective functions are separable means the variants of the CMA-ES with constraints on the covariance matrix are more likely to perform better than the standard CMA-ES on these functions.

As expected, block-CMA-ES performs better than CMA-ES by a factor between one and twenty depending on the function considered, the factor being larger when the ill-conditioning is larger as well. The search costs for the $[1, \ldots, 1]$-block-CMA-ES algorithm are proportional to the dimension. The variants of block-CMA-ES perform better as the number of blocks increase. When the dimension is large, the performances of the fixed block-size variants of block-CMA-ES behave like those of the $[1, \ldots, 1]$-block-CMA-ES. To the opposite, the performances of the fixed number of blocks variants behave like those of the CMA-ES.

When considering the performances versus dimensionality, CMA-ES scales close to linearly on the axis-parallel cigar function and quadratically on the axis-parallel ellipsoid, tablet, two-axes and diffpow functions. The $[n/2, n/2]$ variant scales like the CMA-ES though it performs slightly better by a factor of around 30% at best. The $[1, \ldots, 1]$-block-CMA-ES scales roughly linearly on all of the separable functions though it performs slightly better on the axis-parallel tablet function by a factor smaller than two. On the axis-parallel ellipsoid function, the search costs of $[1, \ldots, 1]$-block-CMA-ES are roughly $300n$ function evaluations. The $[2, \ldots, 2]$ variant performs very similar to the $[1, \ldots, 1]$-block-CMA-ES.

The performances of block-CMA-ES on the axis-parallel functions worsen as the condition number increases. Again the variants of the block-CMA-ES perform better as the number of blocks increase. The performances of the $[1, \ldots, 1]$-block-CMA-ES scale linearly with the condition number.

On the Rotated Version of Separable Functions Instead of the dashed lines, it is now the solid lines in the Figures 3.7, 3.8 and 3.9 that are of concern.

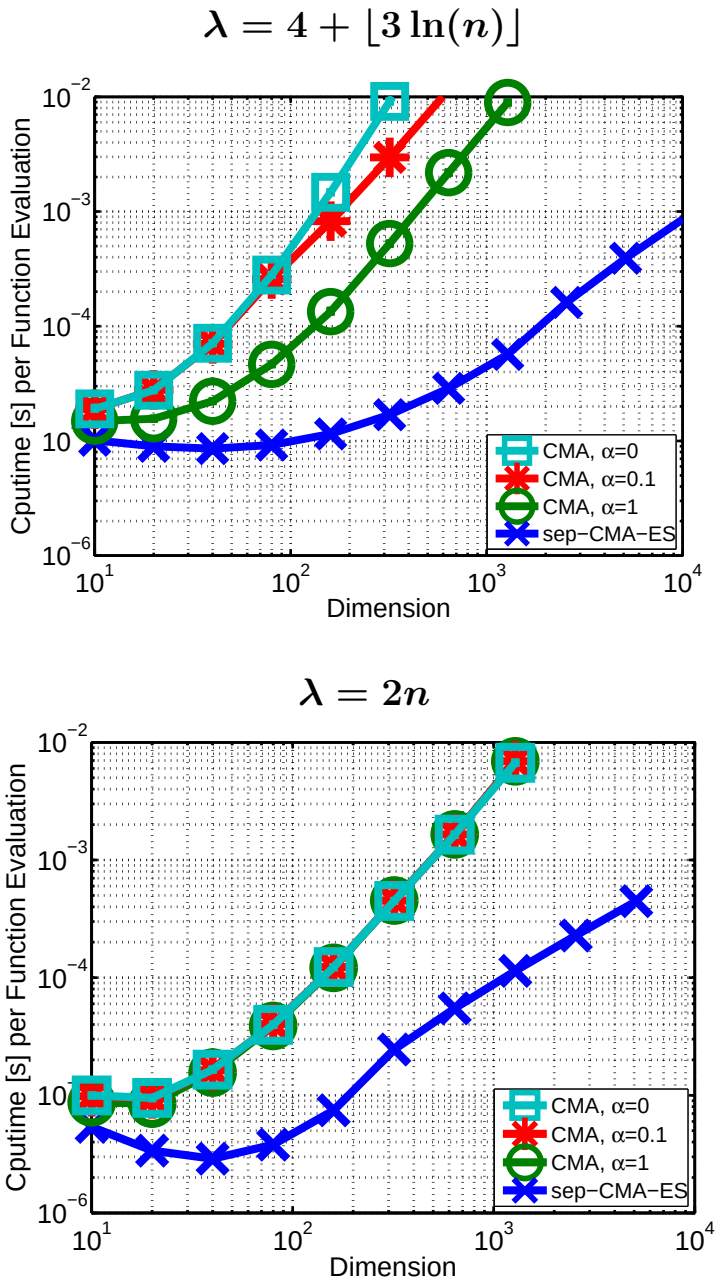


Figure 3.6: CPU-time per function evaluations in seconds of sep-CMA-ES on the axis-parallel ellipsoid function, compared to CMA-ES. Both algorithms are tested for two different population sizes λ. For the CMA-ES algorithm, the eigendecomposition of the covariance matrix is postponed until $(c_{\text{cov}} n)^{-1} \alpha$, α being a parameter. For all experiments, the processing time was measured on three trials of 10^4 evaluations each. Lines show median of the distribution, vertical error-bars show minimum and maximum divided by the number of function evaluations (all indistinguishable).

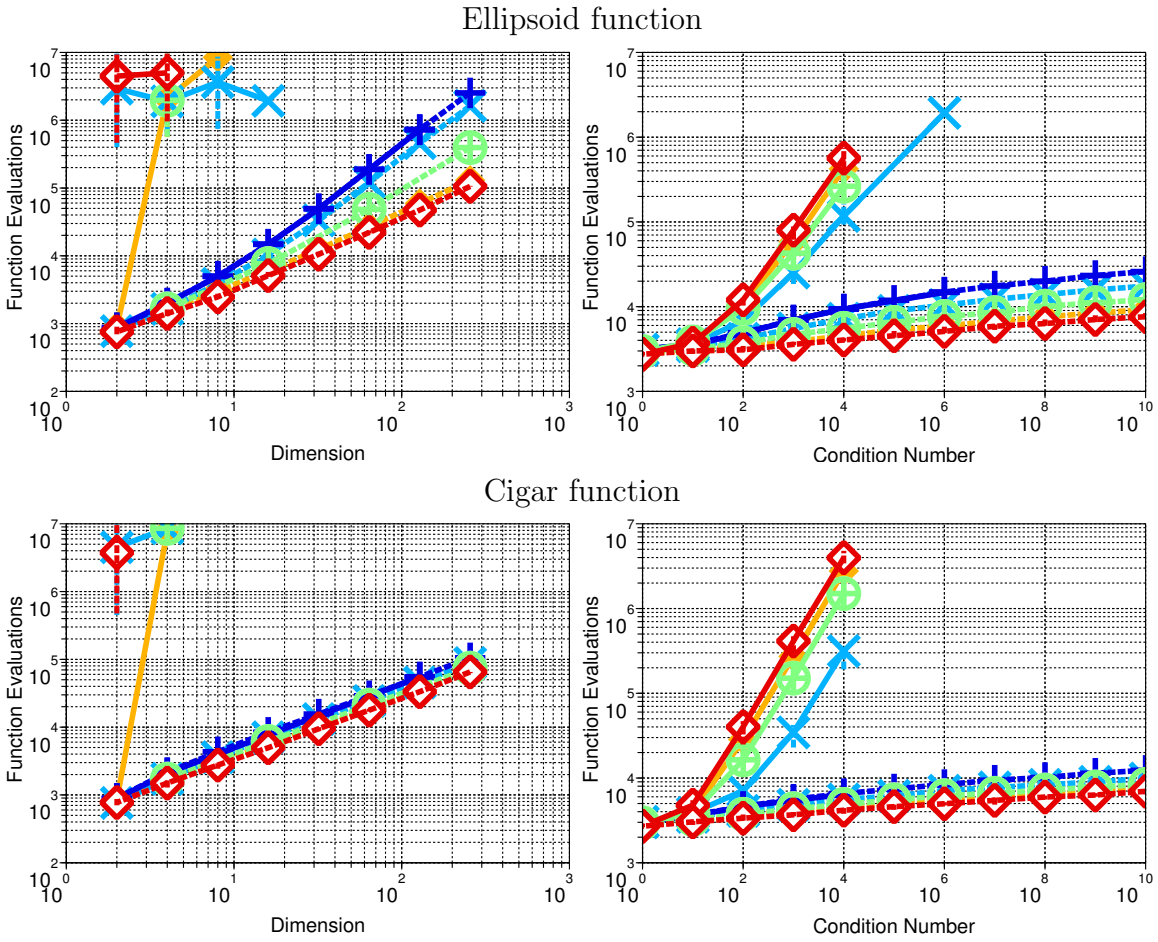


Figure 3.7: Number of function evaluations for reaching a target function value of 10^{-9} for CMA-ES (blue +), and block-CMA-ES with different block configurations: $[1, \ldots, 1]$ (red empty diamonds), $[2, \ldots, 2]$ (orange filled-diamonds), $[\sqrt{n}, \ldots, \sqrt{n}]$ (green circles), $[n/2, n/2]$ (light-blue X), where n is the dimension of the search space, on the ellipsoid and cigar functions with a condition number of 10^6 on the left sub-figures and in 16-D on the right sub-figures. Results for the rotated and axis-parallel functions are represented in solid and dashed lines respectively. Lines show median of the distribution, vertical error-bars show minimum and maximum number of evaluations on successful runs out of eleven runs, only three when the dimension is larger than 100-D.

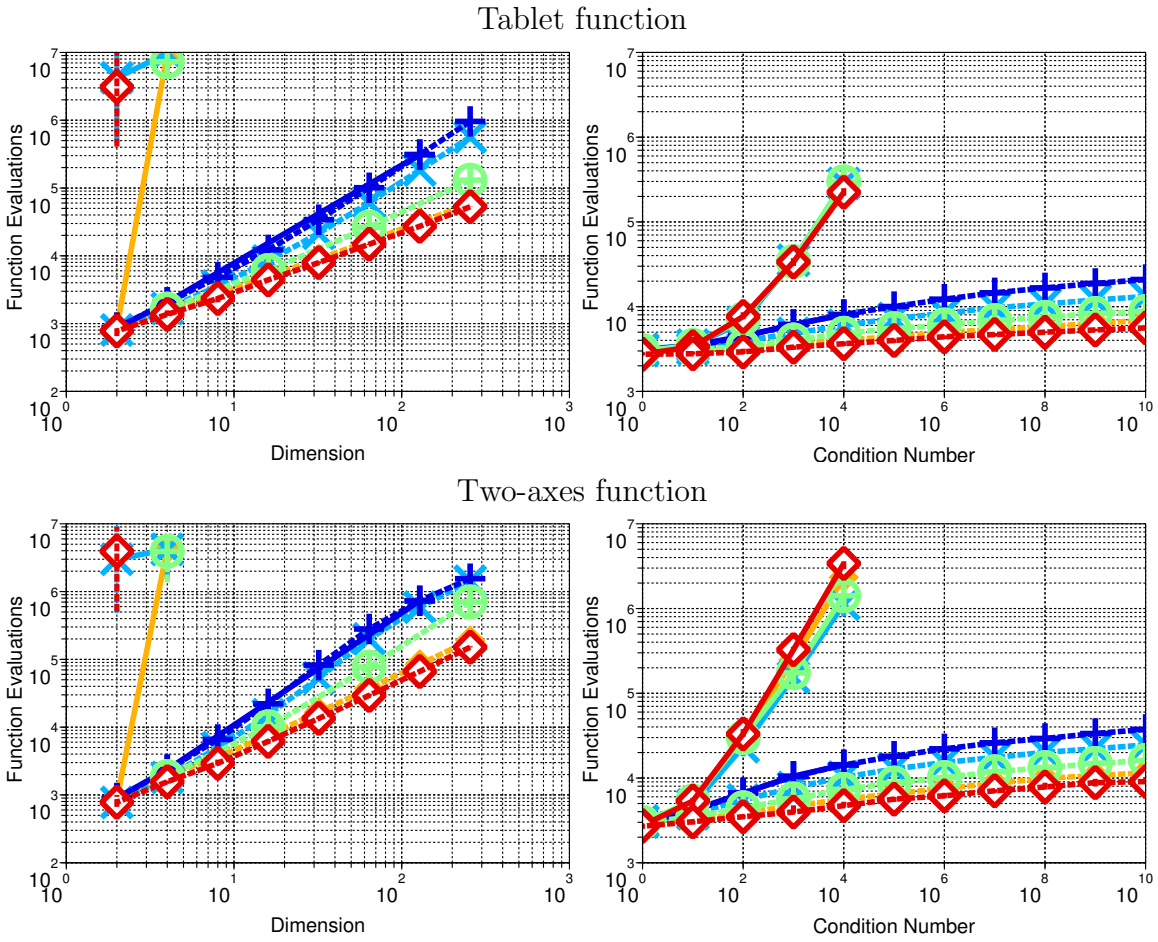


Figure 3.8: Number of function evaluations for reaching a target function value of 10^{-9} for CMA-ES (blue +), and block-CMA-ES with different block configurations: $[1, \ldots, 1]$ (red empty diamonds), $[2, \ldots, 2]$ (orange filled-diamonds), $[\sqrt{n}, \ldots, \sqrt{n}]$ (green circles), $[n/2, n/2]$ (light-blue X), where n is the dimension of the search space, on the tablet and two-axes functions with a condition number of 10^6 on the left sub-figures and in 16-D on the right sub-figures. Results for the rotated and axis-parallel functions are represented in solid and dashed lines respectively. Lines show median of the distribution, vertical error-bars show minimum and maximum number of evaluations on successful runs out of eleven runs, only three when the dimension is larger than 100-D.

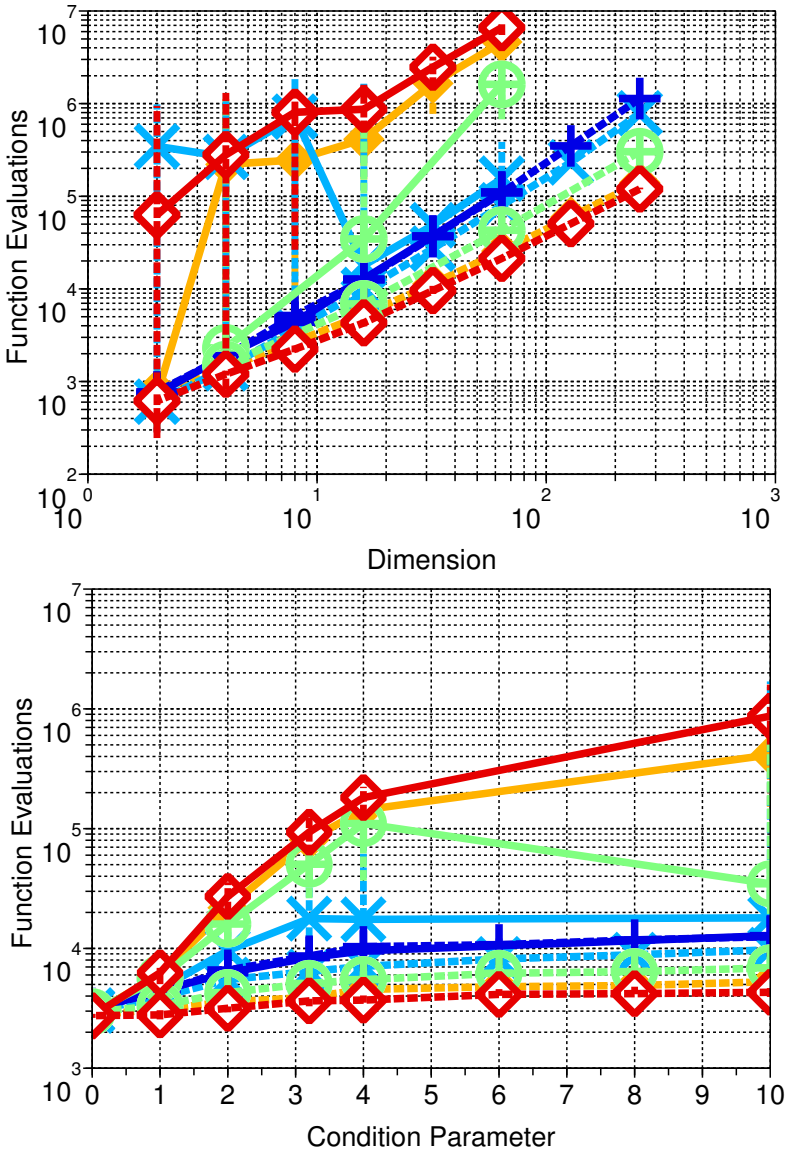


Figure 3.9: Number of function evaluations to reach a target function value of 10^{-14} for CMA-ES (blue +), and block-CMA-ES with different block configurations: $[1, \ldots, 1]$ (red empty diamonds), $[2, \ldots, 2]$ (orange filled-diamonds), $[\sqrt{n}, \ldots, \sqrt{n}]$ (green circles), $[n/2, n/2]$ (light-blue X), where n is the dimension of the search space, on the *diffpow function* with a condition parameter of 10 (top) and in 16-D (bottom). Results for the rotated and axis-parallel functions are represented in solid and dashed lines respectively. Lines show median of the distribution, vertical error-bars show minimum and maximum number of evaluations on successful runs out of eleven runs, only three when the dimension is larger than 100-D.

The performances of block-CMA-ES decrease when the dimension increases on rotated functions. The result is that the block-CMA-ES with more than two blocks does not solve the rotated ellipsoid, cigar, tablet and two-axes functions for dimensions strictly larger than four. Actually, the performances of block-CMA-ES quickly worsen as the condition number increases as well since we can see that in 16-D the same four rotated functions cannot be solved in less than 10^7 function evaluations for a condition number larger than 10^5.

The block-CMA-ES solves the rotated diffpow function for dimensions as large as 64-D. The search costs grow when the dimension is increasing at the exception of the $[n/2, n/2]$-block-CMA-ES to which we will come back right afterwards. We can see the search costs of the $[2, \ldots, 2]$-block-CMA-ES jump up when the problem goes from 2-D to 4-D. This corresponds to going from the $[n]$ block configuration in 2-D to the $[n/2, n/2]$ in 4-D. The performances of the $[n/2, n/2]$ block configuration greatly decrease from 8-D to 16-D. The performances in dimensions smaller than 8-D are comparable to those of the fixed block-size variants whereas the performances are comparable to that of the CMA-ES for dimensions larger than 16-D.

On the Rosenbrock Function The results of block-CMA-ES on the Rosenbrock function are presented in Figure 3.10. On the axis-parallel Rosenbrock function (dashed lines), block-CMA-ES performs worse than CMA-ES until the dimension is large enough. Furthermore, the search costs of the $[n/2, n/2]$ block configuration variant decrease as the dimension increases until 8-D. Then, the search costs of block-CMA-ES increases with the dimension and compares with those of CMA-ES. The search costs of the $[n/2, n/2]$ variant become comparable to that of the CMA-ES in 8-D, those of the $[\sqrt{n}, \ldots, \sqrt{n}]$ variant in 16-D. The search costs of the $[2, \ldots, 2]$ variant are less than three times larger to those of the CMA-ES in dimensions smaller than 32-D, at which point the search costs become equivalent. The $[1, \ldots, 1]$-block-CMA-ES performs the worst in smaller dimension but ends up performing better than CMA-ES by a factor of two when the dimension is larger than 128-D.

In 16-D when the condition parameter increases, the search costs increase. The performances of the $[n/2, n/2]$ variant are comparable to those of the CMA-ES in the range of the condition parameter that we tested. The other variants perform increasingly worse, resulting in the consecutive search costs of the $[1, \ldots, 1]$-block-CMA-ES the $[2, \ldots, 2]$, the $[\sqrt{n}, \ldots, \sqrt{n}]$ and finally the $[n/2, n/2]$ block configurations being

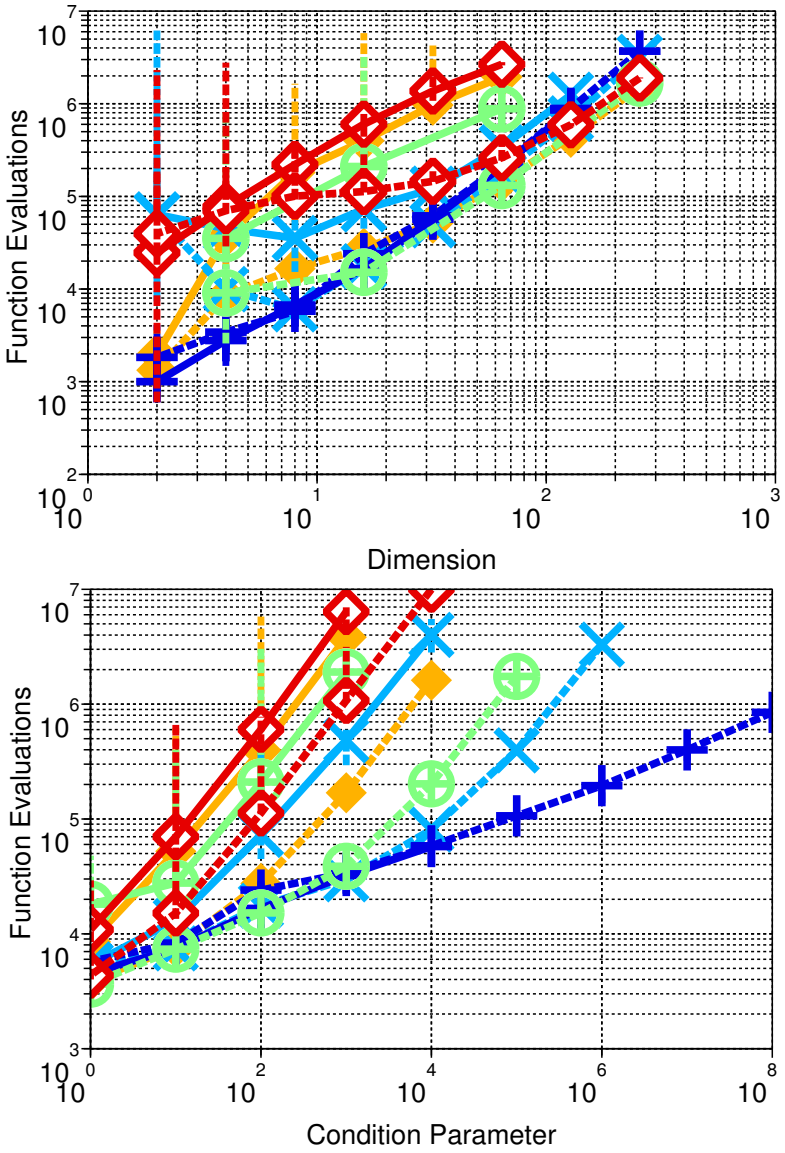


Figure 3.10: Number of function evaluations to reach the target function value of 10^{-9} for CMA-ES (blue +), and block-CMA-ES with different block configurations: $[1, \ldots, 1]$ (red empty diamonds), $[2, \ldots, 2]$ (orange filled-diamonds), $[\sqrt{n}, \ldots, \sqrt{n}]$ (green circles), $[n/2, n/2]$ (light-blue X), where n is the dimension of the search space, on the *Rosenbrock function* with a condition parameter of 100 (top) and in 16-D (bottom). Results for the rotated and axis-parallel functions are represented in solid and dashed lines respectively. Lines show median of the distribution, vertical error-bars show minimum and maximum number of evaluations on successful runs out of eleven runs, only three when the dimension is larger than 100-D.

around five times larger than the next when the conditioning parameter is 10^4.

On the rotated Rosenbrock function (solid lines), we see block-CMA-ES is outperformed by the CMA-ES in all the dimensions that we tested though the overall behaviour seems to point out that this might not be true for dimensions larger than 1000-D. The performances decrease with the number of blocks considered. This behaviour observed when the dimension is larger than 4-D explains the 2-D case where the performances of the $[n/2, n/2]$ block configuration are better than those of the $[2, \ldots, 2]$ variants. In 2-D, $[n/2, n/2]$ block configuration is equivalent to the $[1, \ldots, 1]$ whereas the $[2, \ldots, 2]$ is actually the $[n]$ block configuration. In 16-D, the performances worsen as the condition parameter increases. The performances of the different block configuration worsen as the number of block increases, the difference in the performances being larger for a larger value of the conditioning parameter.

Whereas the performances on the axis-parallel and rotated functions showed comparably large differences on the previous functions, this is not the case on the Rosenbrock function. For instance when the dimension is smaller than 32-D, $[1, \ldots, 1]$-block-CMA-ES on the axis-parallel Rosenbrock function does not perform better than the other variants on the rotated Rosenbrock function. Again, this is also the case in 16-D for different values of the conditioning parameter.

Comparison to Other Algorithms The comparison we provide mainly concerns the sep-CMA-ES. In the comparison in Table 3.2 with indi-ES and in Table 3.3 with AII-ES, we can see that sep-CMA-ES performs as well, otherwise better than the two simpler ES on separable functions such as the axis-parallel ellipsoid, hyper-ellipsoid and diffpow functions. The gain obtained from having a larger population size than for the ES is equal to about 20% for the ellipsoid function, almost 50% for the diffpow function since we also tested sep-CMA-ES with population size (1, 10)-selection.

Comparisons in Table 3.3 and 3.4 show that sep-CMA-ES essentially does better than both MVA-ES and L-CMA-ES, though on the rotated ellipsoid, these two algorithms perform the same as on the axis-parallel ellipsoid whereas the performance of sep-CMA-ES degrades.

On the Rosenbrock function, in the dimensions considered, sep-CMA-ES does not perform well. A (1, 10)-selection policy is better than $(\mu/\mu_W,\ \lambda)$ by a factor of about 30%.

Table 3.2: Comparative performances in terms of number of function evaluations to reach the given target function value, f_{target}. Results in a column are normalised by the best (in bold) for which we give the performance. The dimension is 30-D. The indi-ES refers to [Hansen et al., 1995], its results are taken from its reference. The notation '(1, 10)-select.' refers to a population size of ten with selection of the best individual at each generation. The initial values of the problem-independent parameters $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)}$ and the $\sigma^{(0)}$ are given in the table. The results shown with standard deviation of the number of function evaluations are averaged over 3 runs.

	$f_{\text{hyperelli}}$ axis-parallel $f_{\text{target}} = 10^{-10}$ $\sigma^{(0)} = 1$	f_{Rosen} axis-parallel $f_{\text{target}} = 10^{-6}$ $\sigma^{(0)} = 0.1$	f_{diffpow}^{n-1} axis-parallel $f_{\text{target}} = 10^{-20}$ $\sigma^{(0)} = 1$
Algorithm	$\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} = (1, \ldots, 1)^T$	$\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} = \mathbf{0}$	$\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} = (1, \ldots, 1)^T$
indi-ES (1, 10)-select.	1.1	1.8	**9 700**
sep-CMA-ES (1, 10)-select.	1.3 ± 2%	1.9 ± 3%	1.9 ± 2%
CMA-ES ($7/7_W$, 14)	2.2 ± 3%	**45 000 ± 2%**	8.1 ± 4%
sep-CMA-ES ($7/7_W$, 14)	**5 900 ± 3%**	2.4 ± 2%	**10 000 ± 3%**

3.6 Summary and Perspectives

We presented simple modifications of the CMA-ES that reduces the $\frac{n^2+n}{2}$ strategy parameters of the original algorithm, n being the search space dimension, to the block diagonal components of the covariance matrix —which depending on the size of the block diagonal matrices can reduce the number of strategy parameters to n— resulting in sep-CMA-ES and block-CMA-ES. Dependencies between variables are not captured in sep-CMA-ES and only partially captured for block-CMA-ES. Just like CMA-ES, sep-CMA-ES and block-CMA-ES can both exploit a large population. The *advantages* of sep-CMA-ES are twofold: 1. it provides with a variant of the CMA-ES with linear time and space complexity allowing the sep-CMA-ES to solve large scale problems, 2. the learning rate for the covariance matrix can be increased by a factor of up to $\frac{n+3/2}{3}$, considerably accelerating the adaptation of axis-parallel distribution ellipsoids. The block-CMA-ES provides with a direct control of its internal time and space through the choice of the block configuration, in addition to the benefit of the increased learning rate.

Table 3.3: Comparative performances in terms of number of function evaluations to reach the given target function value, $f_{\text{target}} = 10^{-9}$ in 20-D. Results in a column are normalised by the best (in bold) for which we give the performance. AII-ES refers to [Hansen et al., 1995], MVA-ES has been compared to the CMA-ES rank-1 in [Poland and Zell, 2001], their results without standard deviation are taken from their references. The results shown with standard deviation of the number of function evaluations are averaged over 3 runs. No success was observed for MVA-ES on the ellipsoid function (in 3.5×10^5 function evaluations).

Algorithm	Population	f_{elli} axis-parallel $\sigma^{(0)} = 1$ $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} = (1, \ldots, 1)^T$	f_{Rosen} axis-parallel $\sigma^{(0)} = 0.1$ $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} = \mathbf{0}$
AII-ES	(1, 10)-select.	2.0	**21 000**
MVA-ES	(1, 10)-select.	no success (maxevals 3.5×10^5)	2.7 ± 50 % (Min-max range, 70 runs)
MVA-ES	$(5/5_I,\ 35)$	no success (maxevals 3.5×10^5)	3.7 ± 45 % (Min-max range, 70 runs)
CMA-ES rank-1	(1, 10)-select.	4.4	1.2
CMA-ES rank-1	$(5/5_I,\ 35)$	11	2.4
CMA-ES rank-1	$(6/6_W,\ 12)$	4.4 ± 0.7%	1.2 ± 2%
CMA-ES rank-μ	(1, 10)-select.	4.6 ± 3%	1.3 ± 8%
CMA-ES rank-μ	$(6/6_W,\ 12)$	3.6 ± 2%	**21 000 ± 1%**
sep-CMA-ES rank-1	(1, 10)-select.	1.3 ± 3%	3.5 ± 6%
sep-CMA-ES rank-1	$(5/5_I,\ 35)$	3.1 ± 10%	9.2 ± 7%
sep-CMA-ES rank-μ	(1, 10)-select.	1.3 ± 4%	3.5 ± 3%
sep-CMA-ES rank-1	$(6/6_W,\ 12)$	1.3 ± 3%	6.2 ± 10%
sep-CMA-ES rank-μ	$(6/6_W,\ 12)$	**5 900 ± 10%**	5.6 ± 1%

Table 3.4: Comparative performances in terms of number of function evaluations to reach given target function value, $f_{\text{target}} = 10^{-14}$, in 30-D. Results in a column are normalised by the best (in bold) for which we give the performance. The L-CMA-ES, whose parameter m controls the complexity of the model, was compared to CMA-ES rank-1 in [Knight and Lunacek, 2007], their results are taken from the reference. The results shown with standard deviation of the number of function evaluations are averaged over 3 runs. The population size is $(7/7_W, 14)$.

Algorithm	f_{elli} axis-parallel $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} \in [-5,5]^n$ $\sigma^{(0)} = 5$	f_{Rosen} axis-parallel $\langle \boldsymbol{x} \rangle_{\text{W}}^{(0)} \in [-2,2]^n$ $\sigma^{(0)} = 2$
L-CMA-ES ($m = 5$)	180	1.1
L-CMA-ES ($m = 15$)	64	1.3
CMA-ES rank-1	6.6	1.3
CMA-ES rank-μ	4.1 ± 1%	**48 000 ± 2%**
sep-CMA-ES rank-1	1.4 ± 4%	5.2 ± 5%
sep-CMA-ES rank-μ	**11 000 ± 3%**	4.0 ± 1%

The block diagonal configurations for the covariance imply block-CMA-ES does not learn all the interdependencies of the objective function, whereas the sep-CMA-ES learns none, which favours sep-CMA-ES and block-CMA-ES on separable functions.

We have studied many different block configuration for block-CMA-ES. We have also modified the algorithms sep-CMA-ES presented in [Ros and Hansen, 2008a] so it is equivalent to the $[1, \ldots, 1]$-block-CMA-ES. The sep-CMA-ES and block-CMA-ES algorithms were evaluated on separable and non-separable test functions by measuring numbers of function evaluations. Results from our experiments on some separable functions show the performances of sep-CMA-ES to scale linearly with the dimension, and those of block-CMA-ES to range between the performances of sep-CMA-ES and those of CMA-ES. We also observed that the sep-CMA-ES outperforms variants of CMA-ES that address the time complexity issue of the CMA-ES in the case of separable functions, just as expected.

The well-known Rosenbrock function exhibits relevant dependencies between the

variables posing no principle obstacle for the block-CMA-ES. The $[2, \ldots, 2]$-block-CMA-ES is about three times slower than CMA-ES in smaller dimensions. The performance difference diminishes with increasing dimension and for dimensions larger than 32-D, the performance of the $[2, \ldots, 2]$-block-CMA-ES becomes comparable to that of the CMA-ES. The same occurs with the sep-CMA-ES which is fifty times slower in smaller dimensions, and for dimensions larger than 128-D becomes faster than the CMA-ES. This effect can be attributed to the given coordinate system: on the *rotated* Rosenbrock function neither sep-CMA-ES nor block-CMA-ES outperformed CMA-ES up to 320-D.

A benefit of the study of sep-CMA-ES is to allow to explicitly measure the benefits and drawbacks from learning dependencies. We can quantify the gain or loss that can be attributed to the ability to adapt the complete covariance matrix in CMA-ES. The sep-CMA-ES allows insightful cross-comparisons with other "separable" algorithms (with only coordinate-wise operations). The application of sep-CMA-ES and block-CMA-ES to *real-world problems* will be advantageous (compared to CMA-ES) on high-dimensional objective functions, which either do not have too intricate dependencies between the decision variables (as it is the case for the Rosenbrock function) or are cheap to evaluate. In the first scenario, sep-CMA-ES and block-CMA-ES need fewer function evaluations if the adaptation of the scaling of variables helps to solve the function. The second scenario favours sep-CMA-ES and block-CMA-ES, since the strategy internal time complexity becomes relevant, where CMA-ES is roughly $\frac{n}{10} + 1$ times slower compared to sep-CMA-ES.

Compared to the sep-CMA-ES, the block-CMA-ES offers a compromise between the fully independent sampling of variables that is provided by the sep-CMA-ES and the full matrix learning of the CMA-ES.

Also from our results, we can define a policy: first sep-CMA-ES is used only for a fraction of the learning time of CMA-ES, then one switches to CMA-ES by using the full covariance matrix update rule and changing back the learning rate until the problem is solved. Unfortunately the switch policy will still be significantly inferior in some cases for instance on the Cigar function —for which sufficient dependencies can be learnt in linear time only with CMA— or the separable sum of different powers function —for which the scaling continuously varies and the fast learning rate of sep-CMA-ES would be beneficial for more than just a fraction of the learning time.

Using sep-CMA-ES might improve the comparatively slow performance in the very beginning of an optimisation run of CMA-ES which we ascribe, at least partly, to the fact that overall step-size and in particular coordinate scaling cannot decrease as fast in CMA-ES as in many other evolutionary algorithms.

Additional experiments to test the switch policy, which are not shown in this thesis, show that the use of the sep-CMA-ES prior to switching to CMA-ES does not have a negative impact on the search costs of the CMA-ES on most of the functions presented in this chapter. Only in our additional experiments on the rotated Schwefel function did we observe such a negative impact.

The axis-parallel ellipsoid function is an instance where the switch policy can clearly improve the search costs of the algorithm since on this function the performances of sep-CMA-ES scales linearly whereas those of CMA-ES scales quadratically. The use of sep-CMA-ES is clearly beneficial during the learning phase which can represent a substantial part of the total search costs. Hence, the policy we defined can be stated as follows: we propose to use the sep-CMA-ES for the first $100n/\sqrt{\lambda}$ iterations of the algorithm, where λ is the population size, before switching to the CMA-ES. This switch policy, which is assumed to perform better than the CMA-ES on separable functions or large scale problems and provide with an acceptable compromise on non-separable functions, is tested in the next part of this thesis.

Testing the switch policy as well as the block-CMA-ES on real-world problems is an appealing perspective. Also, some thinking needs to be put in the choice of block configurations depending on the knowledge of the real-world problem considered. In terms of perspectives, the implementation of sep-CMA-ES and block-CMA-ES also serves the purpose of paving the road for variants of the CMA with other types of constraints on the covariance matrix.

Chapter 4

Black-Box Optimisation Benchmarking

Contents

4.1 Introduction

As stated before, our work takes place in the context of continuous unconstrained global optimisation where we consider the black-box scenario, which we referred to as Black-Box Optimisation (BBO). In the context of BBO, *benchmarking* can provide with decisive elements for choosing a solver when facing a new BBO problem. This is assuming that the new BBO problem considered share some similarities with problems of the benchmark. Also, benchmarking may help identifying some very specific weaknesses of algorithms with respect to function properties such as separability or multi-modality. Benchmarking is closely related to experimental analysis

of algorithms, as opposed to the theoretical *worst-case* and *average-case* analyses. Instances of BBO benchmarkings on real-world problems exist, see [Anile et al., 2005, Fowler et al., 2008]. The design of a benchmark needs to implement good practices in experimental analysis, see for instance [Johnson, 2002].

A substantial effort in the implementation of a rigourous BBO benchmarking has been made with the series of IEEE CEC special sessions which proposed standard test suites as well as experimental set-ups for the comparison of algorithms. The CEC special sessions has occurred every year since 2005 to address different fields of optimisation such as multi-objective or dynamic optimisation. Only the CEC 2005 special session on real-parameter optimisation [Hansen, 2006b, Suganthan et al., 2005] addresses the BBO problem that we are concerned with. Since no test suite of real-world problems can possibly cover the whole range of difficulties encountered in BBO, the choice has been made in [Suganthan et al., 2005] to make use of artificial test functions featuring a set of properties that are known to make optimisation difficult.

Some of these difficulties are highlighted here:

dimensionality relates to the growth of the volume of the search space, which happens to be exponential with the dimension of the problem, it also affects the behaviour of algorithms in terms of time and space complexity,

non-separability, learning correlations between the variables is important for the optimisation of the objective function,

multi-modality, the objective function has a number of local minima misleading algorithms in the search of the global optimum,

ruggedness could be considered as highly multi-modal, the landscape of the search space is highly irregular, that is to say non-smooth,

non-convexity can affect methods which make the assumption of convexity such as trust-region methods or gradient-based methods,

ill-conditioning, qualifies in practice functions for which the conditioning is larger than 10^5 , the conditioning being defined as the range over a level set of the maximum improvement of objective function values in a ball of small radius centred on the given level set,

noise adds a certain level of uncertainty in the evaluation of the function and may also add some ruggedness to the objective function involved.

These properties can be used to characterise a given objective function, making it possible to study how the performances of algorithms is affected by these properties either separately or altogether.

The performances of algorithms will be quantified in this chapter by the notion of *search costs*. The search costs, which we have already defined, are the number of function evaluations for the algorithms to surpass a target function value.

Insofar as some types of solvers are usually not benchmarked together, see Chapter 2, we are interested in the comparison of methods from the fields of operational research, deterministic mathematics and evolutionary computation. Methods address the difficulties of optimisation using different ways such as increasing the population size and using restarts to address multi-modality or rescaling the parameters to address ill-conditioning. Some algorithms even have invariance properties which are invaluable for a search algorithm in the sense that it allows to generalise the behaviours of said algorithm to a class of problems, as discussed in [Hansen, 2006a]. We would like to remark that invariance is a priori not associated with good performances. Discussing the invariance properties or the lack thereof of a method is substantial.

First, we designed a reasonably-sized testbed of functions which could offer a way of delving into the behaviours of a few selected algorithms emphasising on the effects of non-separability, non-convexity and ill-conditioning [Auger et al., 2009a,b]. Section 4.2 provides details on the algorithms that we benchmarked. Their benchmarking is presented in Section 4.3.

To extend our results to more algorithms, we provided BBO practitioners with an automated procedure for benchmarking algorithms. A new testbed featuring more function properties was provided. The whole effort resulted in the BBO Benchmarking (BBOB) 2009 workshop that took place during the GECCO 2009 conference, Montreal, Canada. We further present our approach used in BBOB 2009 in Section 4.4, provide and discuss our comparison results. Finally, an overall summary is provided in Section 4.5.

4.2 Algorithms

We introduce some of the algorithms tested in the following sections. We have tested a number of algorithms which originate from diverse branches of continuous optimisation: the Broyden Fletcher Goldfarb Shanno (BFGS) method, the NEW Unconstrained Optimisation Algorithm (NEWUOA), the Differential Evolution (DE) algorithm, the Particle Swarm Optimisation (PSO) algorithm, the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) algorithm and the Monte Carlo search.

4.2.1 BFGS

The BFGS method [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970] is a quasi-Newton method, see Section 2.1.1.1 that iteratively approximates the Hessian of the objective function and proceeds with a local line search. The line search proceeds with an approximation of the Newton direction available by updating $\boldsymbol{B}^{(g)}$ using the BFGS formula:

$$\boldsymbol{B}^{(g+1)} = \boldsymbol{B}^{(g)} - \frac{\boldsymbol{B}^{(g)}\boldsymbol{s}^{(g)}\boldsymbol{s}^{(g)^T}\boldsymbol{B}^{(g)}}{\boldsymbol{s}^{(g)^T}\boldsymbol{B}^{(g)}\boldsymbol{s}^{(g)}} + \frac{\boldsymbol{y}^{(g)}\boldsymbol{y}^{(g)^T}}{\boldsymbol{y}^{(g)^T}\boldsymbol{s}^{(g)}}$$

where $\boldsymbol{s}^{(g)} = \boldsymbol{x}^{(g+1)} - \boldsymbol{x}^{(g)}$, and $\boldsymbol{y}^{(g)} = \nabla f(\boldsymbol{x}^{(g+1)}) - \nabla f(\boldsymbol{x}^{(g)})$.

4.2.2 NEWUOA

The New Unconstrained Optimization Algorithm (NEWUOA) [Powell, 2006] is a trust region method, see Section 2.1.1.2, that iteratively updates the internal model of the objective function.

A new quadratic model is built after the step from $\boldsymbol{x}^{(g-1)}$ to $\boldsymbol{x}^{(g)}$ based on a minimum-change updating process. The model is interpolated from q interpolation points where q ranges from $n+2$ to $\frac{1}{2}(n+1)(n+2)$ and the remaining degrees of freedom are taken up by the minimisation of the Frobenius norm of the variation in the second-derivative matrix of the model. Compared to the generic trust region method presented in Section 2.1.1.2, many more refining techniques are used in the management of the trust region radius and the update of the quadratic model.

The initial and final values of the radius of the trust region are also parameters

of the algorithm.

4.2.3 PSO

The Particle Swarm Optimization (PSO) algorithm, see Section 2.1.2.4, has many different variants. Here we are interested in the particular instance of the Standard PSO 2006[1]. The default size of the swarm is $S = 10 + \lfloor 2\sqrt{n} \rfloor$. Each particle has access to $\boldsymbol{p}_i$, their own previous all-time best position, and to $\boldsymbol{g}_i$, the position of the "global best". If the global best did not improve at the previous step, the "global best" $\boldsymbol{g}_i$ is determined from the previous best of the subset of the i-th particle itself and K particles randomly drawn with replacement in the whole swarm, meaning there could be from zero to K different particles in addition to the particle itself to choose among. The velocities and positions are updated as follows:

$$\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + c_1\boldsymbol{r}_2 \circ (\boldsymbol{g}_i - \boldsymbol{x}_i) + c_2\boldsymbol{r}_2 \circ (\boldsymbol{p}_i - \boldsymbol{x}_i)$$
$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are random vectors which elements are uniformly drawn in the range $[0, 1]$, $\circ$ is the element-wise multiplication. The default values of the parameters are: $K = 3$, $w = \frac{1}{2\ln 2}$, $c_1 = c_2 = \frac{1}{2} + \ln(2)$. Out of bounds particles see their position set to the boundaries and their velocity set to zero.

4.2.4 DE

Differential Evolution (DE) is an evolutionary algorithm that makes use of a differential mutation which basically adds the weighted difference between two population vectors to a third vector, see Section 2.1.2.8. Many variants of the differential mutation procedure exists. Choosing between these variants and setting the parameter F and CR required preliminary tests since [Storn and Price, 1997] admits that the results of the algorithm are dependent on the chosen strategy and the choice of parameter.

[1]The C-code is available at: http://www.particleswarm.info/Standard_PSO_2006.c

4.2.4.1 Identification of the Parameters of DE

We test the original code provided by Storn[2] which proposes six strategies [Price et al., 2005]. The DE/x/y notation specifies for x the vector to be mutated, y the number of difference vectors used. The way the strategies are numbered is relevant since it corresponds to the numeration in the code.

1. DE/rand/1 is the original DE as presented in Section 2.1.2.8,
2. DE/local-to-best/1 is a variant where instead of the base vector $\boldsymbol{x}_{i_1}$ being chosen in the population vector, it is chosen to lie between the vector considered and the best vector so far, thus the mutant candidate is written as follows: $\boldsymbol{v}_i = \boldsymbol{x}_i + F(\boldsymbol{x}_{\text{best}} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{i_2} - \boldsymbol{x}_{i_3})$,
3. DE/best/1 with jitter, in this case the base vector is the best vector so far, a quantity uniformly sampled between zero and 10^{-4} is added to the parameter F for the computation of each component of $\boldsymbol{v}_i$, different for each vector $\boldsymbol{v}_i$,
4. DE/rand/1 with per-vector-dither is the original DE with a quantity uniformly sampled between zero and $1 - F$ added to F, different for each vector $\boldsymbol{v}_i$,
5. DE/rand/1 with per-generation-dither is the original DE with a quantity uniformly sampled between zero and $1-F$ added to F, different at each generation,
6. DE/rand/1 either-or algorithm is the original DE but using randomly either the classical differential mutation or a three-point-recombination.

All six strategies are tested on the rotated ellipsoid function with a condition number of a hundred in 5, 10 and 20-D, see Table 4.1 for its analytical expression. The starting point of the algorithm is uniformly sampled in the range $[-3, 10]^n$. The population sizes considered are 1, 3, 5, 10 and 30 times the dimension of the search space. The maximum number of function evaluations is set to $1000n$ times the population size. The parameter CR is chosen in the range $[0, 1]$, F in $[0.3, 0.9]$. Each experiment is repeated three times.

The performance measure is the average number of function evaluations to reach the target function value 10^{-7}.

[2] MATLAB code available here: http://www.icsi.berkeley.edu/~storn/code.html

4.2.4.2 Results and Discussion of the Identification of the Parameters of DE

Only the results of DE with the default population size of ten times the dimension of the search space are shown here in Figures 4.1, 4.2 and 4.3. All results of our parameter identification experiments can be found in Appendix A. The number of function evaluations for DE to reach the target function value depends on the dimensionality of the problem and the values of F and CR considered. The ratio between the performances the best and worst settings can be as large as a thousand. Overall the best performances are obtained with CR and F both close to one, though it is worth noting that the number of function evaluations does not behave monotonically as for a given CR when F increases and for a given F when CR increases.

These preliminary tests led us to consider the DE/local-to-best/1/bin variant, also denoted as DE/target-to-best/1/bin in [Price et al., 2005] which uses a single difference between a random vector and the best-so-far vector and uniform cross-over with $F = 0.8$ and $CR = 1$ and default population size.

4.2.5 CMA-ES

The CMA-ES algorithm is a stochastic, population-based search method in continuous search spaces, see Chapter 3. We used the $(\mu/\mu_W, \lambda)$-CMA-ES [Hansen and Kern, 2004].

4.2.6 Monte Carlo Search

We compare the previous algorithms to the Monte Carlo search, see Section 2.1.2.1, a baseline algorithm where the search space is uniformly sampled.

4.3 Study on Non-Separability, Ill-Conditioning and Non-Convexity

In this section, we present our benchmarking of the effect of non-separability, ill-conditioning and non-convexity on the performances of some state-of-the-art algorithms by providing some well-known benchmark functions.

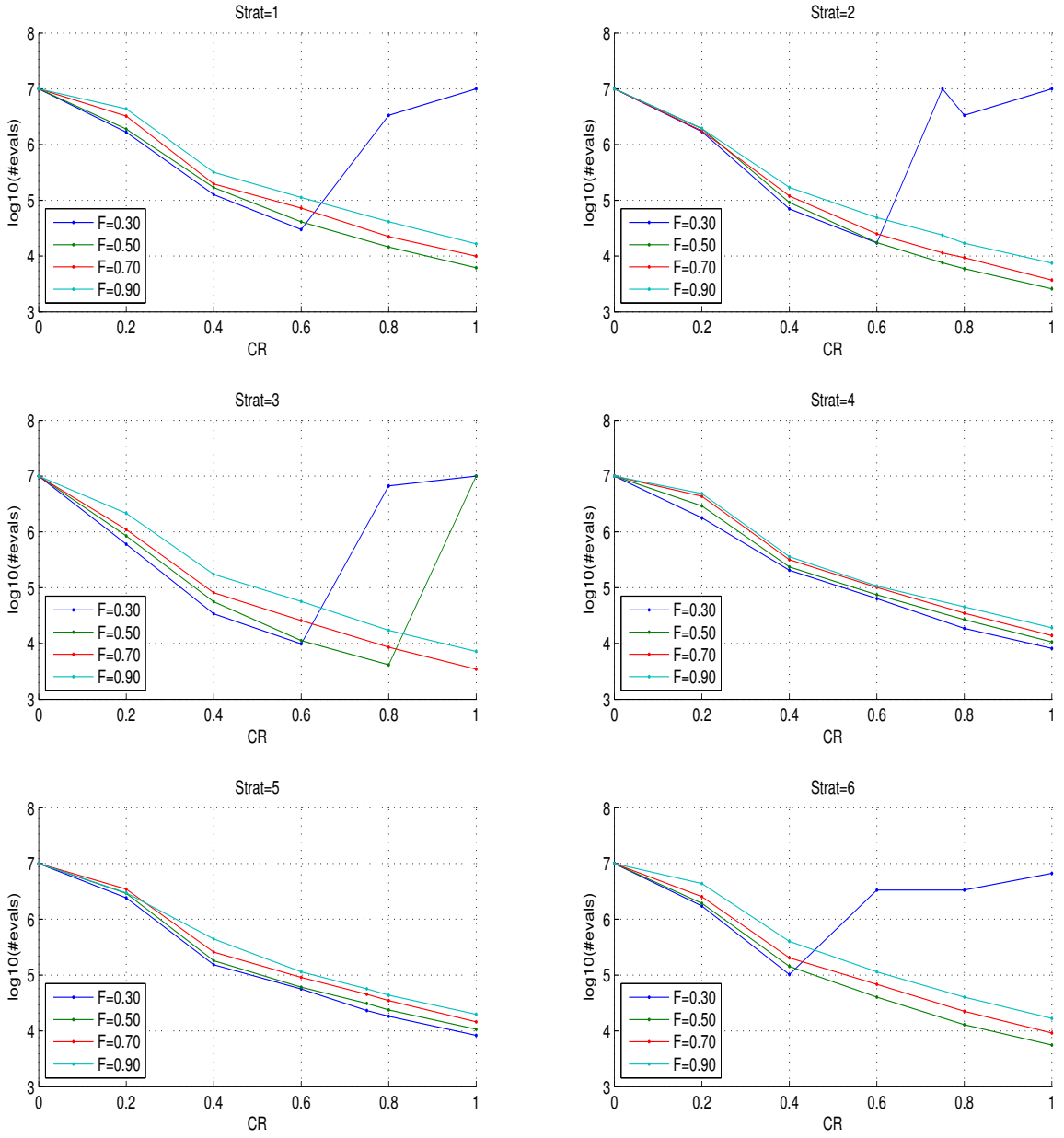


Figure 4.1: Identification of the parameters of DE on the rotated ellipsoid function in 5-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

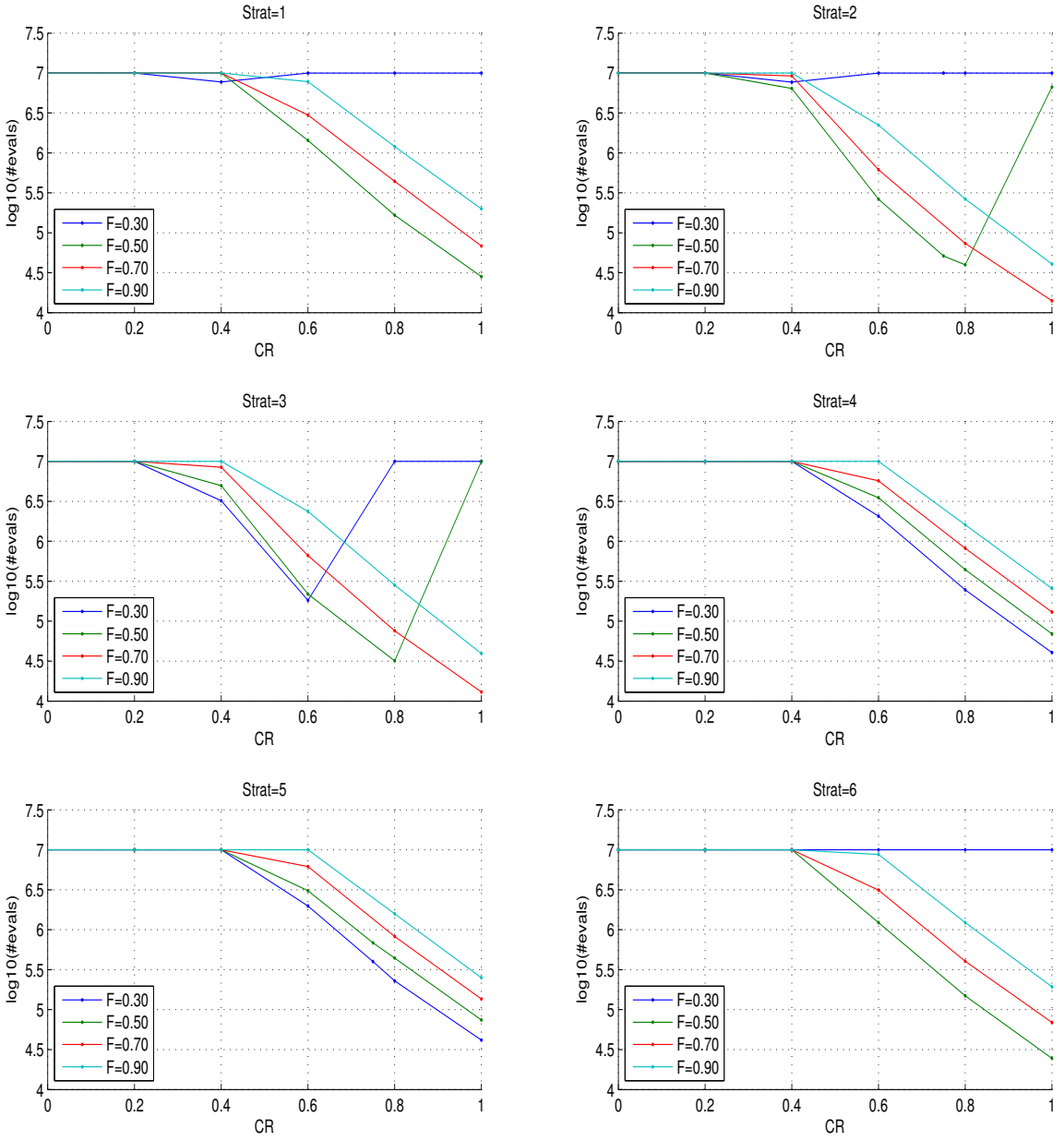


Figure 4.2: Identification of the parameters of DE on the rotated ellipsoid function in 10-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

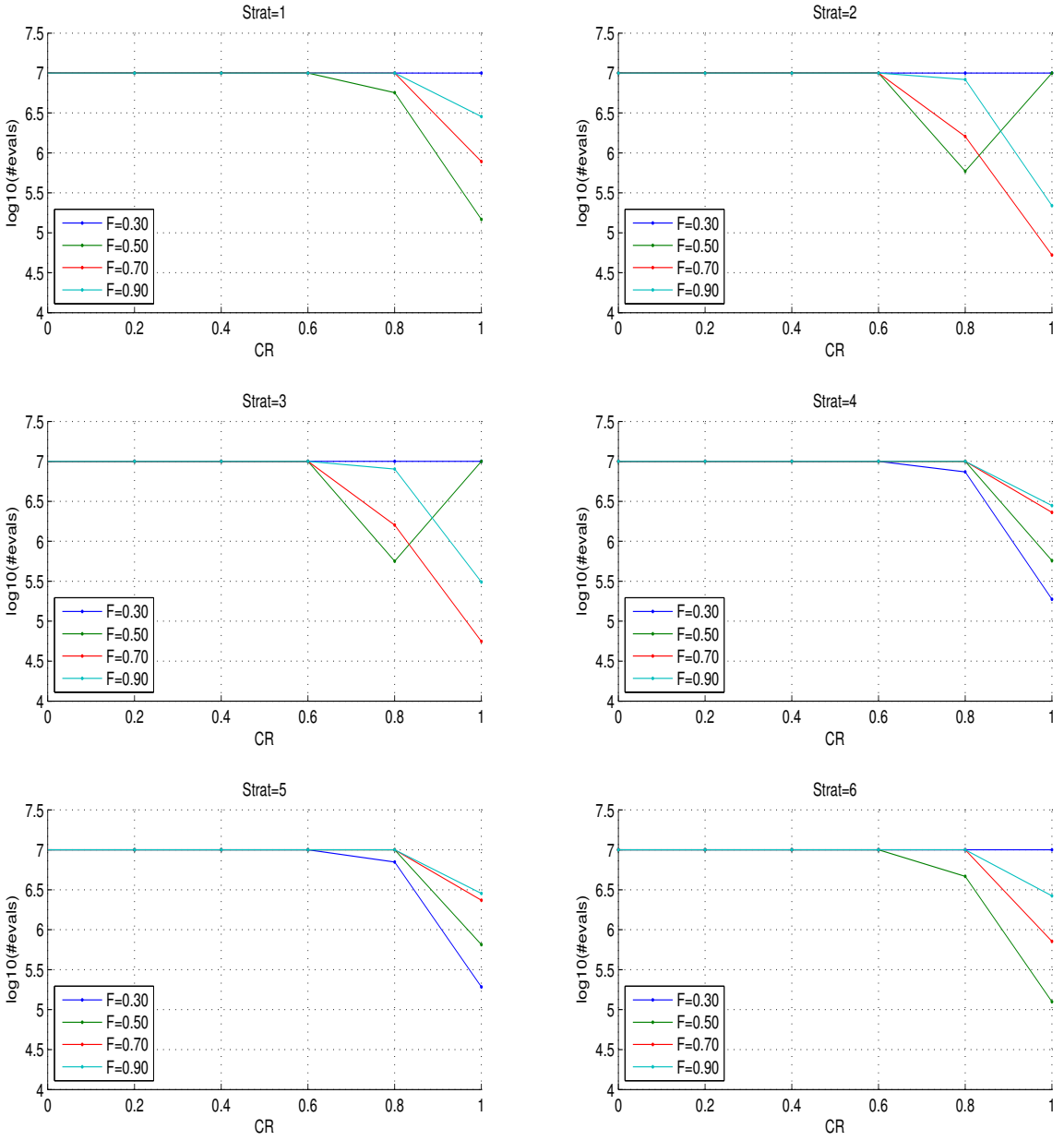


Figure 4.3: Identification of the parameters of DE on the rotated ellipsoid function in 20-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

Table 4.1: Test functions with coordinate-wise initialisation intervals and target function value, where $\boldsymbol{y} := \boldsymbol{Q}\boldsymbol{x}$ implements an angle-preserving, linear transformation, *i.e.* $\boldsymbol{Q}$ is orthogonal.

Function	β	Initialisation	f_{target}
$f_{\text{elli}}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta^{\frac{i-1}{n-1}} y_i^2$	$[1, 10^{10}]$	$[-20, 80]^n$	10^{-9}
$f_{\text{Rosen}}(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left(\beta \left(y_i^2 - y_{i+1}\right)^2 + (y_i - 1)^2\right)$	$[1, 10^{8}]$	$[-20, 80]^n$	10^{-9}
$f_{\text{elli}}^{1/4}(\boldsymbol{x}) = \left(\sum_{i=1}^{n} \beta^{\frac{i-1}{n-1}} y_i^2\right)^{1/4}$	$[1, 10^{10}]$	$[-20, 80]^n$	10^{-9}

The benchmark consisted of the ellipsoid and Rosenbrock functions. The DE algorithm, the PSO algorithm, the BFGS method, the CMA-ES algorithm and the NEWUOA, all described in Section 4.2, are benchmarked. We detail our test functions and methods in Section 4.3.1, then provide and discuss our results in Section 4.3.2 and Section 4.3.3.

4.3.1 Test Functions and Methods

We present the test functions and the methods in the subsequent sections.

4.3.1.1 Test Functions

The functions (see Table 4.1) are tested in their original axis-parallel version (*i.e.* $\boldsymbol{Q}$ is the identity and $\boldsymbol{y} = \boldsymbol{x}$), and in rotated versions, where $\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{x}$. The orthogonal matrix $\boldsymbol{Q}$ is chosen such that each column is uniformly distributed on the unit hypersphere surface [Hansen and Ostermeier, 2001], fixed for each run.

The ellipsoid function f_{elli} is a convex-quadratic function which optimum value is zero obtained at the origin of the search space. The parameter β is the condition number of the Hessian matrix that is varied between one and 10^{10} in our experiments. If the condition number is equal to one the ellipsoid is the isotropic separable sphere function. The function $f_{\text{elli}}^{1/4}$ has the same contour lines (level sets) as f_{elli}, however it is neither quadratic nor convex. For $\beta \neq 1$, the functions f_{elli} and $f_{\text{elli}}^{1/4}$ are separable if and only if $\boldsymbol{Q} = \boldsymbol{I}$.

The Rosenbrock function f_{Rosen} is non-separable, has its global minimum value zero at $\boldsymbol{x} = [1, 1, \ldots, 1]$ and, for large enough β and n, has one local minimum close

to $\boldsymbol{x} = [-1, 1, \ldots, 1]$ [Shang and Qiu, 2006]. The contour lines of the Rosenbrock function show a bent ridge that guides to the global optimum (the Rosenbrock is sometimes called banana function) and the parameter β controls the width of the ridge. In the classical Rosenbrock function, β equals 100. For smaller β, the ridge becomes wider and the function becomes less difficult to solve. We vary β between one and 10^8.

These functions were altered with respect to: 1. non-separability by considering rotation of the search space, 2. ill-conditioning by considering the conditioning as a parameter, and 3. non-convexity by applying the composition to the left of the objective function with the square root of square root function which results in a non-convex function.

4.3.1.2 Implementation of Benchmarked Algorithms

We thereafter describe the experimental set-up for the algorithms. For the BFGS method, a MATLAB implementation was used. It is accessible using the generic function `fminunc` (revision 1.1.6.3) that proposes, among others, the BFGS method for the update of the Hessian Matrix. The stopping criteria are set so runs stop only when line search fails due to round-off errors.

As for the NEWUOA, the implementation used for our experiments is the one provided by Matthieu Guibert[3] which delivers Powell's original FORTRAN source code of the algorithm [Powell, 2006]. Minor changes have been brought to this code to fit with our experimental set-up. For our experiments, we considered for q the number of interpolation points the value $2n + 1$ (recommended by Powell [2006]) where n is the dimensionality of the problem. The initial radius of the trust region was set to a hundred, corresponding to the length of the initialisation range, see below. The stopping criterion which is the final radius of the trust region is set to 10^{-15}. The maximum number of function evaluations was set to 10^8.

The implementation of PSO that we used is the standard PSO 2006[4] translated into SCILAB[5]. The neighbourhood parameters are all set to their default values.

Also for the CMA-ES, the SCILAB implementation (version 0.92)[6] was used. The

[3] http://www.inrialpes.fr/bipop/people/guilbert/newuoa/newuoa.html
[4] Available at this location: http://www.particleswarm.info/Programs.html
[5] For which we acknowledge Nikolas Mauny.
[6] Latest version available here: http://www.lri.fr/~hansen/cmaesintro.html

initial step-size was set to 10/3. All other parameters are set to their default values.

For DE, the DE/local-to-best/1, second strategy available in the MATLAB code provided by Price[7] was used. The cross-over constant CR was set to one and the weighting factor F to 0.8. The population size was chosen to be ten times the dimension of the search space.

4.3.1.3 Methods and Performance Measures

For each algorithm tested we conduct twenty-one independent trials of up to 10^7 function evaluations. For all algorithms, initial points have been sampled uniformly in the range $[-20, 80]^n$. For BFGS, if no success was encountered, the number of trials was extended to a thousand and one. We quantify the performance of the algorithms using the probability of success p_S and the success performance SP1 and used in [Hansen and Kern, 2004], analysed in [Auger and Hansen, 2005]. The probability of success is the ratio of the successful runs over the total number of runs. The SP1 equals the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations. The SP1 is an estimator of the expected number of function evaluations to reach the target function value if the algorithm is restarted until a single success (supposing infinite time horizon) and assuming that the expected number of function evaluations for unsuccessful runs equals the expected number of evaluations for successful runs. The fact that SP1 is computed after a given number of runs can result in a high variance of the values of SP1 in the case of a small ratio of successful runs. Another possibility, which has not been exploited in this manuscript, would be to compute SP1 for a given number of successful runs.

4.3.2 Results

We compare the performances of the algorithms on the test functions that we presented in terms of SP1. We provide only rough figures for the comparisons we make here since, to draw our conclusions, we are more interested in differences that are represented by orders of magnitude.

The performances of all algorithms worsen as the condition number increases, see

[7] Available at: http://www.icsi.berkeley.edu/~storn/code.html

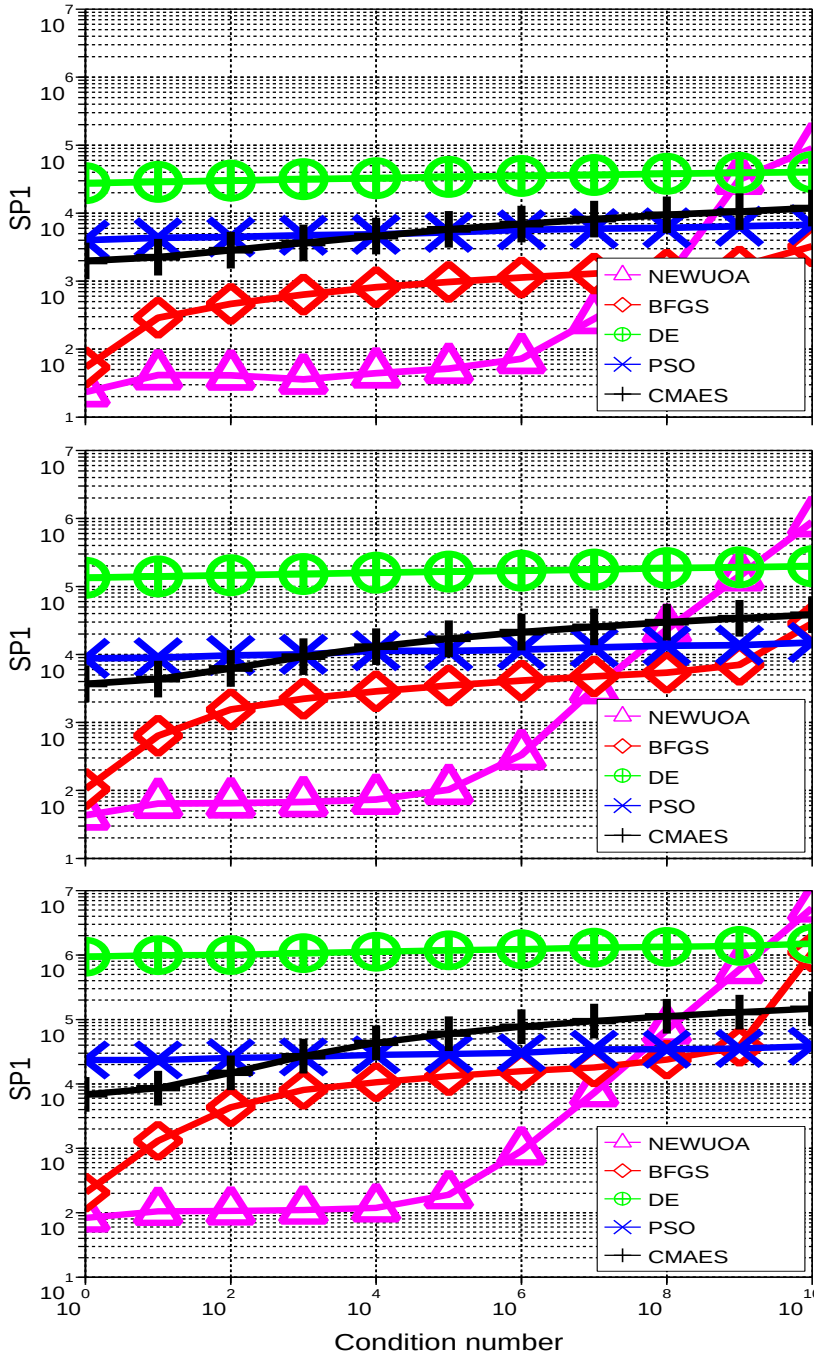


Figure 4.4: Effects of the ill-conditioning of the axis-parallel ellipsoid function on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition number of the objective function.

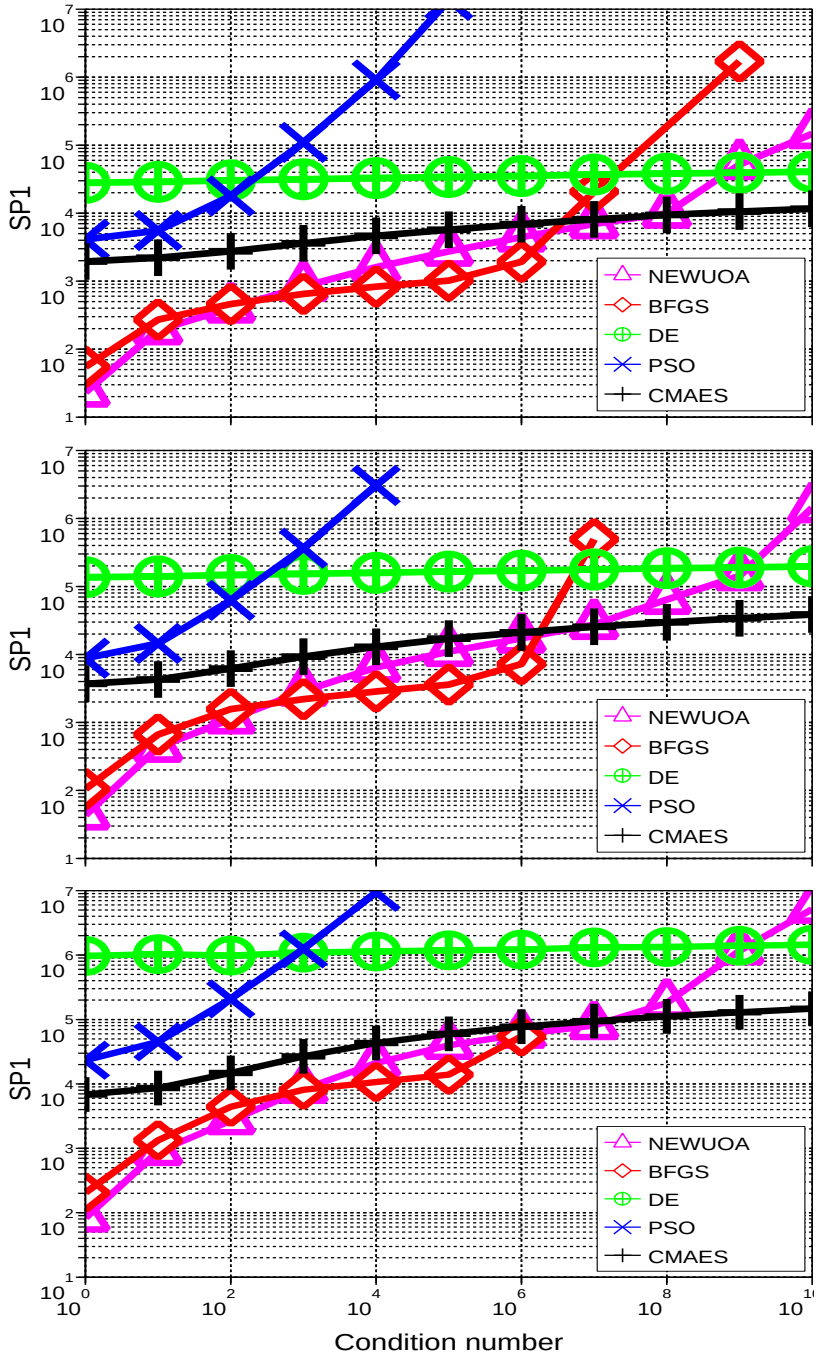


Figure 4.5: Effects of the ill-conditioning of the rotated ellipsoid function on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition number of the objective function.

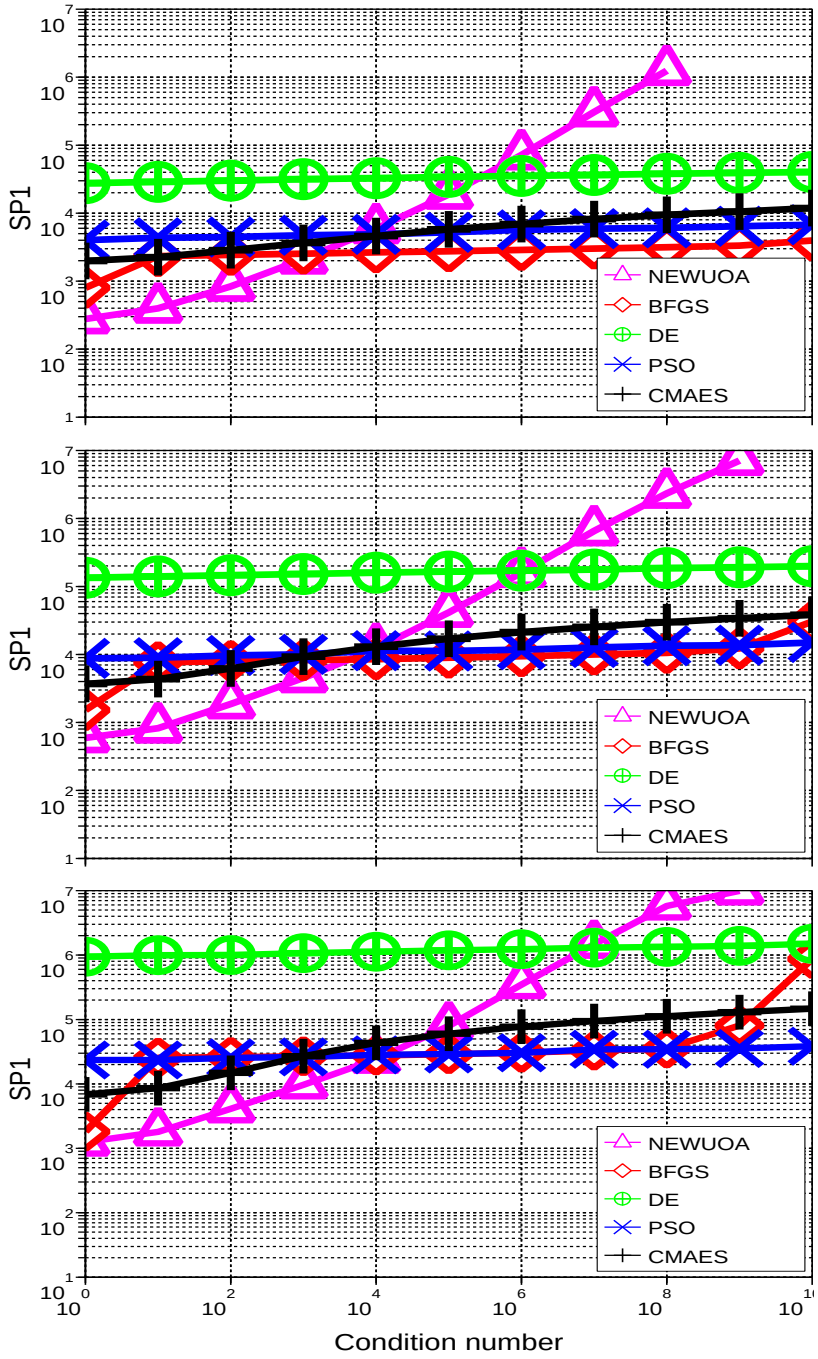


Figure 4.6: Effects of the ill-conditioning of the axis-parallel ellipsoid to the power one fourth on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition number of the original ellipsoid function.

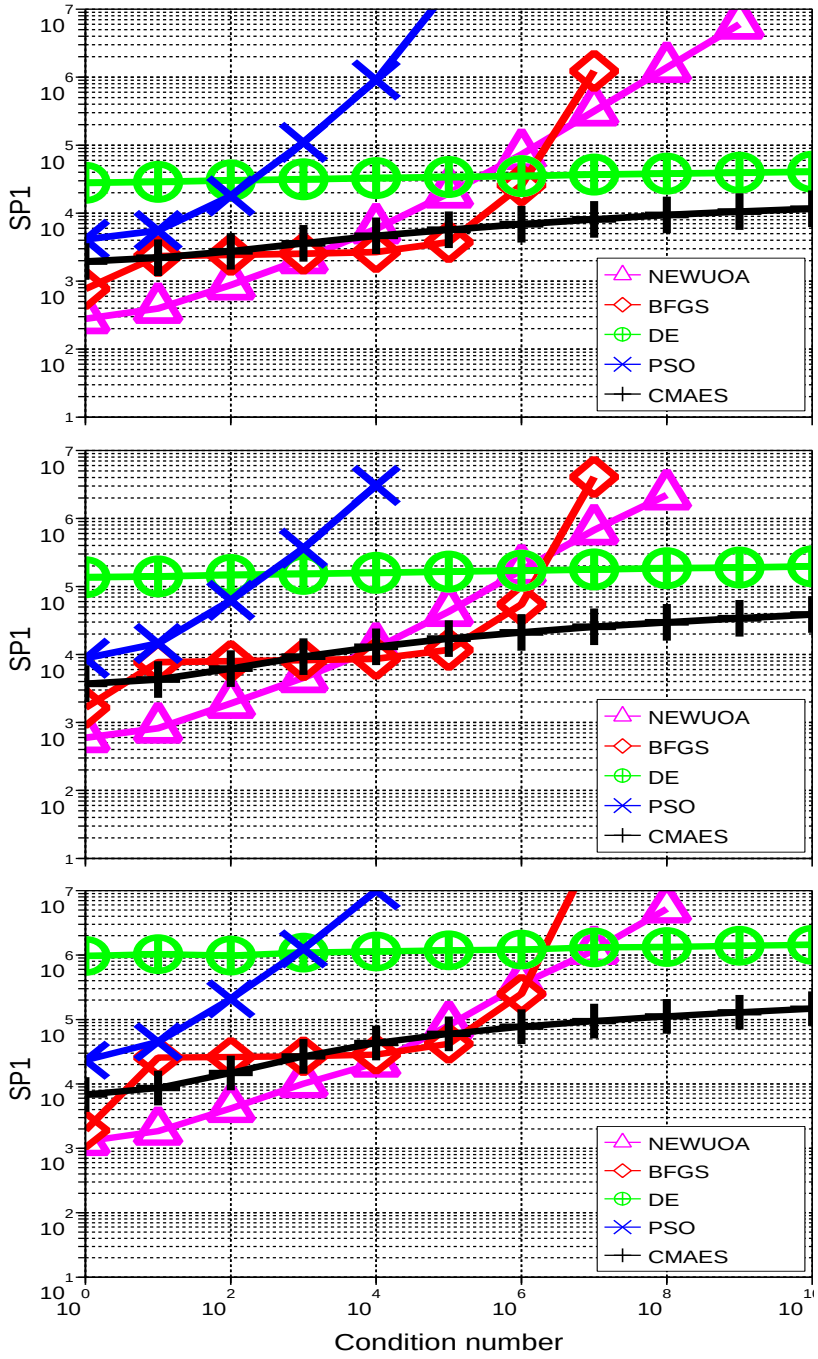


Figure 4.7: Effects of the ill-conditioning of the rotated ellipsoid to the power one fourth on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition number of the original ellipsoid function.

Figures 4.4 to 4.9. And as expected the performances of all algorithms worsen as the dimension increases, this effect being more pronounced for larger condition numbers.

On the Axis-Parallel Ellipsoid Function, see Figure 4.4, the BFGS method, the NEWUOA and the CMA-ES algorithm are more affected by the increasing condition number: the SP1 increasing by a factor of around ten thousand, a hundred and ten respectively when the condition number goes from one to 10^{10}. This factor increases from 10-D to 40-D by up to an order of magnitude, two for the BFGS method. The BFGS method is particularly affected for condition numbers larger than 10^8 due to numerical round-off errors in the computation of the gradient using a finite-difference method. In contrast, the increase in the SP1 of PSO and DE is not larger than a factor of two.

A factor of around six for PSO, ten for CMA-ES, ten for BFGS, thirty for DE, sixty for NEWUOA can be observed for the increase of SP1 from 10-D to 40-D for a condition number of 10^8, this factor increasing by an order of magnitude only for BFGS when the condition number is larger.

As for comparisons, for a condition number smaller than 10^8, the performances of NEWUOA are better than those of BFGS by a factor of up to a hundred for a condition number of 10^4 on the axis-parallel ellipsoid function. Nevertheless, the SP1 of NEWUOA outgrows that of BFGS and finally all algorithms given that the condition number is larger than 10^9. For condition numbers ranging from one to 10^7 both NEWUOA and BFGS are better than CMA-ES and PSO, the next best algorithms. The SP1 of CMA-ES and PSO is larger than that of NEWUOA by a factor of one to two hundred for a condition number smaller than 10^6. The performances of CMA-ES and PSO are close, with CMA-ES being the best out of the two when the condition number is smaller than 10^3, and PSO faring better by a factor of up to three otherwise. The PSO algorithm is actually the best algorithm for a condition number of 10^{10} in 20 and 40-D. The performances of DE always come last.

The success probability of CMA-ES, DE and NEWUOA on the twenty-one trials is a hundred percent, that of PSO is larger than 95%. The success probability of BFGS is close to a hundred percent except for condition numbers close to 10^{10} where the success probability can drop as low as 2% on the thousand and one trials.

On the Rotated Ellipsoid Function, see Figure 4.5, the performances of CMA-ES or DE are not affected. The performances of the BFGS method worsen for condition numbers larger than 10^6, the reason being that the finite-difference computations lead to numerical round-off errors, which are more likely to occur in the case of the rotated ellipsoid. The performances of the NEWUOA worsen by a factor of up to two hundred for condition numbers ranging from ten to 10^7. The performances of the NEWUOA for larger condition numbers are comparable on the rotated and axis-parallel ellipsoid function. The PSO algorithm is particularly affected since we can see that the SP1 depending on the condition number increases dramatically faster than for the axis-parallel ellipsoid function. This results in the PSO algorithm not reaching the optimum in less than 10^7 function evaluations on the ellipsoid function for a condition number of 10^5 which can barely be described as large.

The performances of NEWUOA are close to those of BFGS up to a condition number of 10^6 before the SP1 of BFGS increases dramatically as described before. The performances of NEWUOA and BFGS are the best out of the performances of the algorithms considered for condition numbers smaller than 10^6; though for a condition number larger than 10^2 the difference between the performances of CMA-ES and the best out of NEWUOA and BFGS is less than a factor of five. The performances of both NEWUOA and BFGS decline for larger condition numbers to become worse than that of CMA-ES and DE which become best and second best for condition numbers larger than 10^9, those of PSO coming last for all condition numbers tested on the rotated ellipsoid function.

The success probability of CMA-ES, DE and NEWUOA is a hundred percent. When the condition number is larger than 10^5, the success probability of both BFGS and PSO is close to a hundred percent before quickly falling to zero.

On the Ellipsoid Function to the Power One Fourth, see Figures 4.6 and 4.7, the performances of the CMA-ES, the PSO and the DE algorithms are not affected. Only the results of the BFGS method and the NEWUOA, both of which assume that the objective function can be approximated by a convex quadratic model, are affected by the non-convexity of the ellipsoid function to the power one fourth. The behaviour of the performances of the BFGS changes only slightly, though the performances worsen by a factor of up to twenty compared to those obtained on the

ellipsoid function. The performances of the NEWUOA decrease by a factor of up to ten and the behaviour changes such that, on the axis-parallel ellipsoid to the power one fourth, for condition numbers smaller than 10^5, the SP1 increases whereas it was close to constant on the axis-parallel ellipsoid. Another effect of the non-convexity on NEWUOA is that the performances of the algorithm is similar on the axis-parallel and rotated ellipsoid function to the power one fourth.

The NEWUOA is still the best algorithm, this time only for a condition number smaller than 10^3. The performances of the BFGS are now close to those of CMA-ES for the rotated function and both CMA-ES and PSO on the axis-parallel function, being only better by a factor of less than two in 10-D, and even less in 20 and 40-D. Nevertheless, the sorting of the performances of the algorithm actually does not change when compared to that of the convex quadratic ellipsoid function, only the ranges of the condition numbers do: on the axis-parallel functions the best algorithm is NEWUOA for smaller condition numbers, then BFGS is until the condition number is large enough so that the performances of PSO make it rank first; on the rotated function, NEWUOA followed by BFGS and then CMA-ES are the best algorithms considering their performances as the condition number increases.

The probability of success of BFGS on the axis-parallel ellipsoid to the power one fourth is close to a hundred percent until the condition number is 10^{10} in which case it drops to 50% in 10 and 20-D and goes to zero in 40-D. In the case of the rotated ellipsoid to the power one fourth, the probability of success drops for a condition number of around 10^6. The probability of success of NEWUOA is close to a hundred percent before dropping to zero.

On the Rosenbrock Function, see Figures 4.8 and 4.9, again the performances of the algorithms worsen when the dimensions of the search space increase, and when the conditioning parameter increases. The rotation of the objective function affects BFGS and PSO. The performances of PSO worsen by a factor of up to ten. On the Rosenbrock, the probability of success of BFGS drops quickly to zero when the conditioning parameter is larger than three hundred. On the rotated Rosenbrock function, this probability of success is larger than for the axis-parallel Rosenbrock function when β is larger than three hundred.

The performances of NEWUOA and CMA-ES ranging from 10 to 40-D decrease by a factor close to four for $\beta = 1$, to ten for $\beta = 10^2$, to twenty for $\beta = 10^6$. The

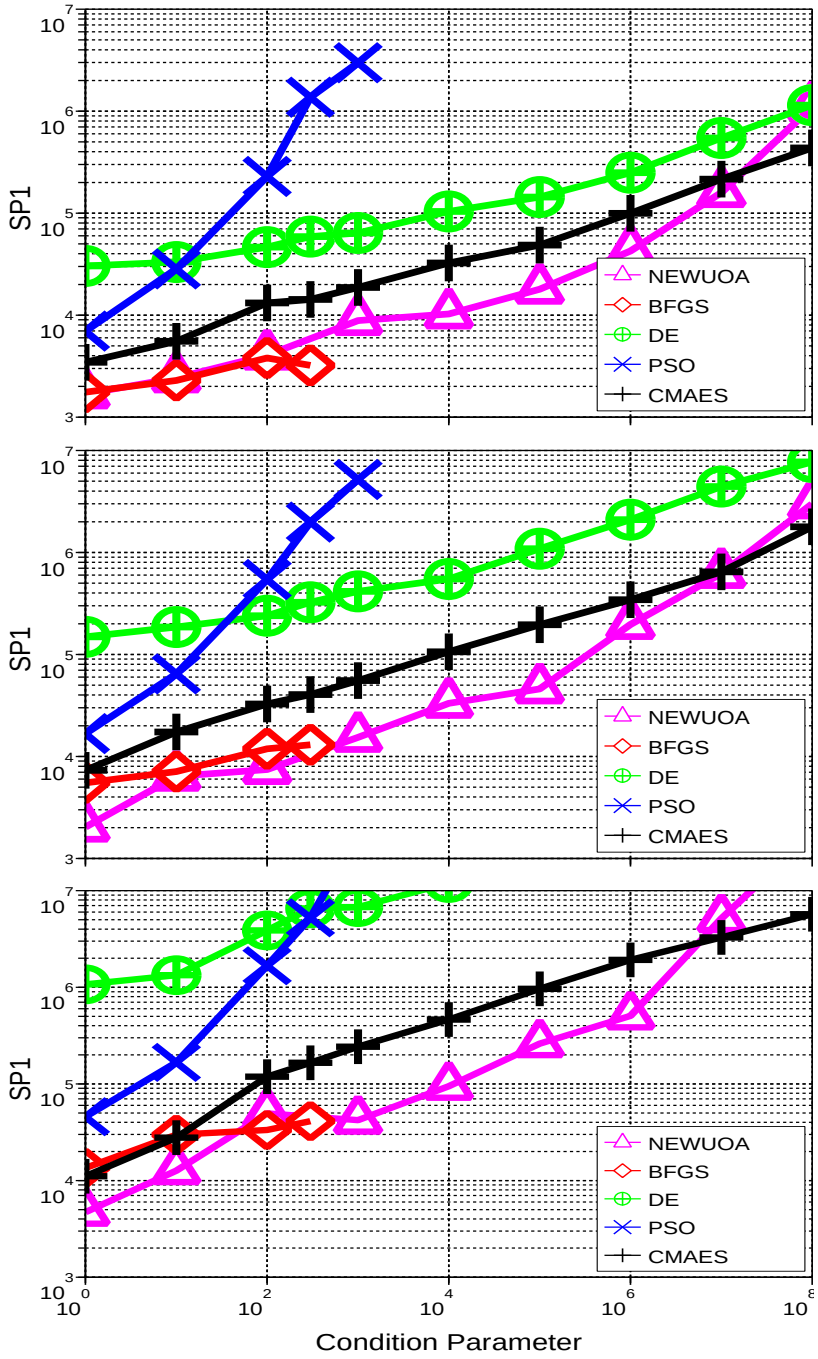


Figure 4.8: Effects of the ill-conditioning of the Rosenbrock function on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition parameter β of the objective function.

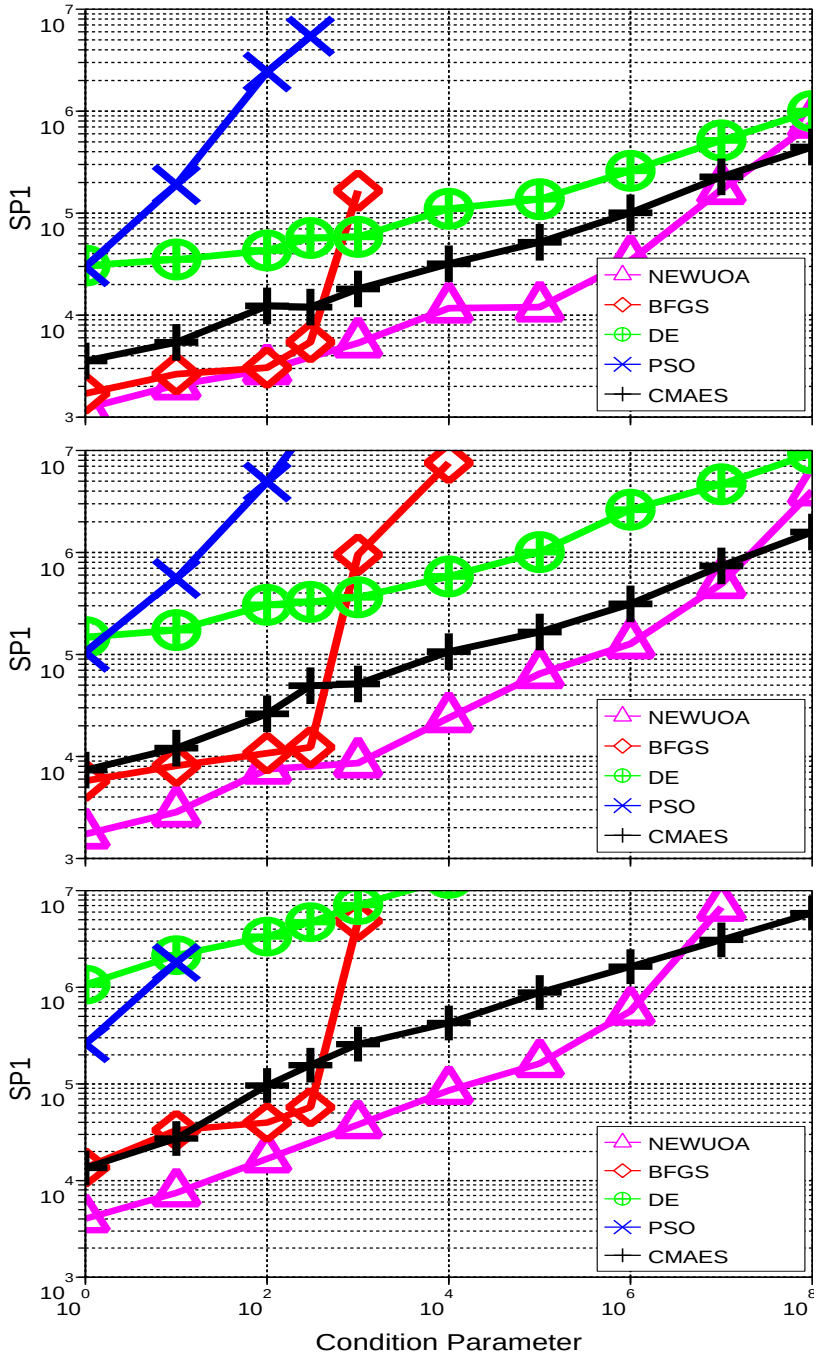


Figure 4.9: Effects of the ill-conditioning of the rotated Rosenbrock function on BFGS, NEWUOA, DE, PSO and CMA-ES in 10, 20, 40-D. Shown is SP1, which is the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations, versus the condition parameter β of the objective function.

performances of BFGS and PSO decrease by a factor close to ten for $\beta = 1$ and $\beta = 10^2$, twenty and a hundred respectively for DE.

In terms of comparison, NEWUOA is the best algorithm by a factor of three at best compared to the performances of the CMA-ES, until the conditioning parameter β is equal to 10^7. The performances of NEWUOA then worsen such that the performances of CMA-ES become best. For a conditioning that is smaller than 10^3, the performances of BFGS are in-between those of NEWUOA and CMA-ES. The performances of PSO, and DE come last. The performances of PSO are behind those of CMA-ES by a factor of around three when β is one, this factor increases quickly as β increases. The SP1 of PSO grows larger than that of DE when β is larger than a hundred.

The probability of success of the CMA-ES, DE and NEWUOA is larger than 70%, 60% and 50% respectively. When the condition parameter β is larger than 10^3 the probability of success of both BFGS and PSO quickly goes to zero.

4.3.3 Summary and Discussion of our Study on Non-Separability, Ill-Conditioning and Non-Convexity

We highlight some of the results presented previously.

4.3.3.1 Results on Invariances

Invariances of algorithms in benchmarking play an important role insofar as an algorithm showing an invariance to a certain function property can generalise its performance to other functions.

In this respect, the algorithms show different kinds of invariances.

PSO and DE are *almost* invariant to conditioning of the ellipsoid function. For PSO, this statement is only true on the axis-parallel ellipsoid function. This statement has a limited impact since this does not extend to the Rosenbrock function. The CMA-ES, BFGS and NEWUOA are comparatively much more affected by the ill-conditioning.

CMA-ES, DE are invariant to rotation of the search space. This is true for DE because the cross-over parameter is set to 1. The CMA-ES is inherently invariant due to its use of the covariance matrix. The NEWUOA is affected by the rotation of the search space on the ellipsoid function with conditioning smaller than 10^5, for which it performs extremely well in the case of the axis-parallel ellipsoid. To the opposite, the Standard PSO 2006 is greatly affected by such search space transformation, see below.

CMA-ES, DE and PSO are invariant to the non-convex transformation. The three algorithms do not make use of the function values but the ranking of the individuals according to these function values to progress. Therefore, a monotonic transformation of the objective function has no effect on the performances of these algorithms.

4.3.3.2 PSO on the Rotational Transformation

The PSO is highly affected by the search space rotation as our results show, see also Figure 4.10. The behaviour of Standard PSO as opposed to that of the CMA-ES with regards to rotational invariance was studied in more details in [Hansen et al., 2009d].

4.3.3.3 NEWUOA and BFGS

The stochastic methods we considered are not affected by the non-convex transformation of the objective function. This is not the case for NEWUOA which performances on the ellipsoid function are extremely good though highly dependent on the function being convex-quadratic. As our results show, the performances of NEWUOA decline comparatively rapidly when considering the ellipsoid to the power one fourth. The performances of BFGS though not nearly as badly affected show some declining as well.

One of the most surprising revelations of our study is presented in the following. Despite that the functions considered are standard test functions of continuous optimisation, the BFGS and NEWUOA merely do better than CMA-ES by a factor of five at best for the rotated ellipsoid function with only slight conditioning or for the

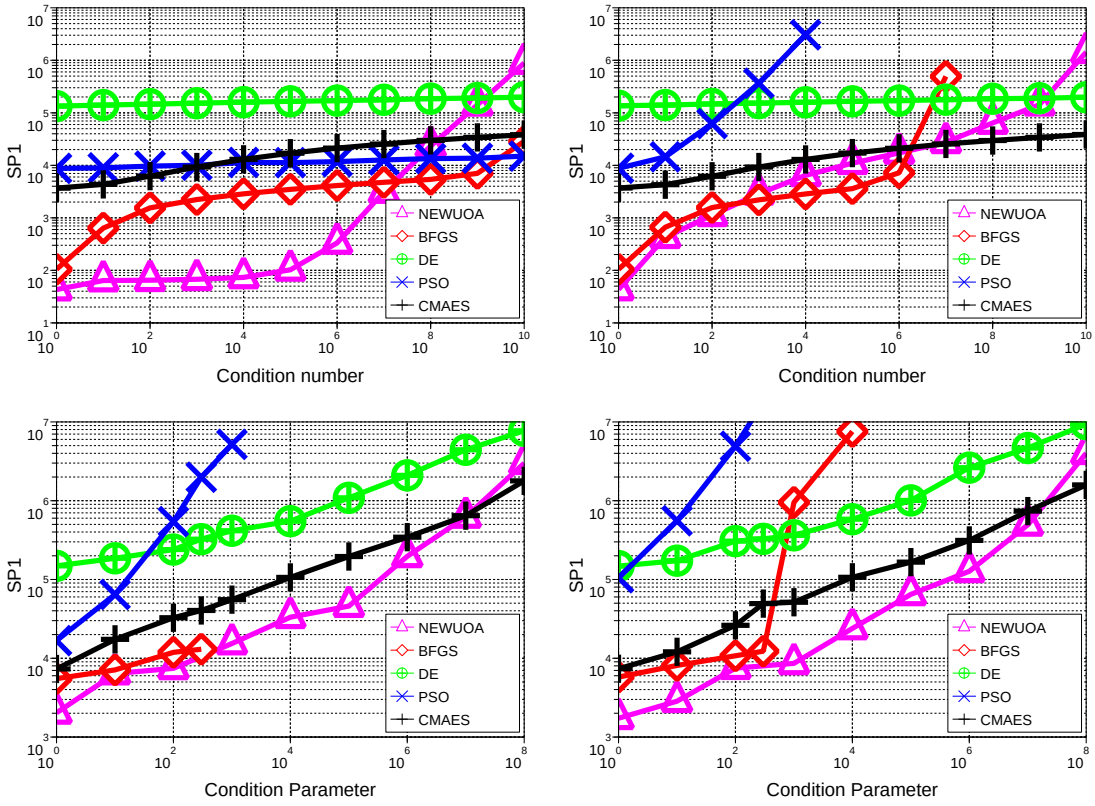


Figure 4.10: Effect of the rotation on the PSO (blue X) and other algorithms depending on the condition number and parameter as seen on the 20-D ellipsoid (top row) and Rosenbrock (bottom row) functions respectively.

Rosenbrock function. This could not be expected especially since the test case considered is actually the best case scenario for a trust region method such as NEWUOA or a line search method such as BFGS.

4.4 BBOB 2009

The Black-Box Optimisation Benchmarking (BBOB) 2009 is a benchmarking event that took place as a workshop in GECCO 2009, Montreal, Canada[8]. The BBOB 2009 featured the COCO software platform[9] as well as a full experimental set-up. The BBOB 2009 resulted in comparison results between algorithms originating from the field of evolutionary computation or deterministic mathematics and operations research. The algorithms were tested on an extended testbed of test functions and an additional testbed of functions using three noise models.

While the technical implementation of the BBOB 2009 is presented in Chapter 5, we only discuss our scientific approach and our results here. On the base of this benchmarking framework, we are also providing reports of our experience as participants since we used the BBOB 2009 to benchmark some algorithms both from a classical background and of our own. The next section provides details of the algorithms that we have benchmarked. The Section 4.4.2 presents the experimental set-up that we designed. Results of our benchmarked algorithms are presented in Section 4.4.3. Results in the light of the overall benchmarking in the BBOB workshop are discussed in Section 4.4.4.

4.4.1 More Benchmarked Algorithms

We have benchmarked the BFGS method, the NEWUOA, the IPOP-sep-CMA-ES and the Monte Carlo search algorithms. Except for the IPOP-sep-CMA-ES, all algorithms were introduced in Section 4.2.

[8] http://www.sigevo.org/gecco-2009/workshops.html#bbob
[9] http://coco.gforge.inria.fr

4.4.1.1 Restarts

In the context of a determined initial budget, the notion of restarts, or multi-start, can improve the probability of the algorithm to reach a target function value [Hansen et al., 2009a]. An independent restart means that the algorithm forgets what was learnt during previous iterations to start anew. The problem of choosing when a restart should occur to optimise the performance of an algorithm is not trivial.

For BFGS and NEWUOA, we add an independent restart procedure that starts the algorithm anew from a random initial point in the search space when the algorithm stops without being successful.

4.4.1.2 IPOP-sep-CMA-ES

The sep-CMA-ES is a variant of the CMA-ES algorithm, presented in details in Chapter 3, exploiting separability. Here we have used the sep-CMA-ES before switching back to the full covariance matrix update rule of the original CMA-ES. All internal parameters (except the learning rate) are retained in the process, among those the acquired diagonal covariance matrix. The desired behaviour is to benefit from the faster learning of the sep-CMA-ES in the case of separable problems and then revert to the original CMA-ES [Hansen and Kern, 2004].

The IPOP restart strategy, introduced in [Auger and Hansen, 2005] was added to the sep-CMA-ES. The IPOP restart strategy consists in a restart procedure that starts the algorithm anew from a random point in the search space, but with a population size that is *doubled*. This improves the success probability of the CMA-ES on multi-modal functions [Auger and Hansen, 2005], but can also result in misguidance in the case of multi-funnels function [Lunacek et al., 2008, Müller and Sbalzarini, 2009].

4.4.1.3 BBOB 2009 Entries

In the context of the BBOB 2009, in addition to BFGS, NEWUOA, IPOP-sep-CMA-ES and Monte Carlo search that we already mentioned but put here for sake of completeness, many more algorithms were proposed by participants of the workshop. The results of the algorithms MCS [Huyer and Neumaier, 1999], SNOBFIT [Huyer and Neumaier, 2008] and GLOBAL [Csendes, 1989] are not published alongside the

others in [Rothlauf, 2009]. We merely list here all algorithms altogether and refer to their workshop papers.

BFGS [Ros, 2009a,b] is a quasi-Newton method, see Section 4.2.1,

NEWUOA [Ros, 2009c,d] is a trust region method, see Section 4.2.2,

IPOP-sep-CMA-ES [Ros, 2009e,f] is the $(\mu/\mu_w, \lambda)$-CMA-ES using the sep-CMA-ES strategy for only the first few iterations with increasing population restarts, see Section 4.4.1.2,

Monte Carlo Search or Pure Random Search [Auger and Ros, 2009a,b] is the algorithm that consists in sampling the search space uniformly, see Section 2.1.2.1,

(1+1)-ES with one fifth success rule [Auger, 2009a,b] is the base evolution strategy, see Section 2.1.2.7,

BayEDA$_{\mathbf{cG}}$ [Gallagher, 2009a,b] is an estimation of distribution algorithm, see Section 2.1.2.9, that uses Bayesian inference to learn the model parameters of the probability density estimation model used,

AMaLGaM IDEA or Adapted Maximum-Likelihood Gaussian Model Iterated Density-estimation Evolutionary Algorithm [Bosman et al., 2009a,b] is an estimation of distribution algorithm, see Section 2.1.2.9, which uses the maximum-likelihood for the mean and covariance matrix,

iAMaLGaM IDEA [Bosman et al., 2009a,b] is the same as the AMaLGaM IDEA with an incremental model building,

(1+1)-CMA-ES [Auger and Hansen, 2009a,b] is the CMA-ES, see Section 3.2, with one offspring, a '+' selection rule, see Section 2.1.2.7, and independent restarts,

BIPOP-CMA-ES [Hansen, 2009a,b] is the $(\mu/\mu_w, \lambda)$-CMA-ES, see Section 3.2, with restarts and a small and large population management strategy,

Cauchy EDA [Posik, 2009a] is an estimation of distribution algorithm, see Section 2.1.2.9, using the Cauchy distribution as probabilistic model,

DASA or Differential Ant-Stigmergy Algorithm [Korosec and Silc, 2009a,b] is an ant-colony approach combined with probability sampling, see Section 2.1.2.3,

G3-PCX or Generalized Generation Gap (G3) model with parent centric crossover (PCX) [Posik, 2009c] is an evolutionary algorithm, see Section 2.1.2.5,

simple GA [Nicolau, 2009] is a genetic algorithm adapted to continuous optimisation by increasing the number of bits to encode each variable, see Section 2.1.2.5,

POEMS or Prototype Optimization with Evolved IMprovement Steps [Kubalik, 2009] is an evolutionary algorithm, see Section 2.1.2.5, that uses so called hypermutations,

EDA-PSO [El-Abd and Kamel, 2009a] introduces a probability to use either an estimation of distribution approach, see Section 2.1.2.9 or the particle swarm optimisation equations, see Section 2.1.2.4 at each iteration,

PSO [El-Abd and Kamel, 2009b] used the global best model with the parameters w, c_1 and c_2 different from zero, see Section 2.1.2.4,

PSO_Bounds [El-Abd and Kamel, 2009c] is based on the PSO, see Section 2.1.2.4, with an additional adaptive bounds procedure and restart strategy,

DEPSO [García-Nieto et al., 2009a,b] is a PSO algorithm, see Section 2.1.2.4, using the differential variation scheme in DE, see Section 2.1.2.8, for adjusting particular velocities,

DIRECT [Posik, 2009b] is an algorithm using partitioning of the search space with hyper-rectangles iteratively divided so as to balance local and global searches, see Section 2.1.1.4,

MCS or Multilevel Coordinate Search [Pál et al., 2009a,b] is inspired by DIRECT with an additional local search procedure, see Section 2.1.1.4,

SNOBFIT or Stable Noisy Optimization by Branch and FIT [Huyer and Neumaier, 2009a] is also related to DIRECT but adapted to noisy objective function by combining the direct search with the maintenance of local surrogate functions, see Section 2.1.1.4,

GLOBAL [Huyer and Neumaier, 2009b,c] looks for the local minima that are potentially global by using a combination of sampling, clustering and local search, see Section 2.1.1.4,

Line Search Methods, [Posik, 2009d] see Section 2.1.1.1 have been tested, one based on golden section search and parabolic interpolation denoted as LSfminbnd, one which is an uni-variate search algorithm based on interval division denoted as LSstep,

Rosenbrock's Local Search Method [Posik, 2009e] maintains a model of the current local neighbourhood and then proceeds with an approach close to that of conjugate-gradient methods,

Nelder-Mead method is a simplex method, see Section 2.1.1.3; two variants were tested: one with a reshaping step in the search and a population-based approach [Doerr et al., 2009] and another with different restarts strategies [Hansen, 2009c],

MA-LS-Chain or Memetic Algorithm using Local Search Chaining [Molina et al., 2009a,b] is a hybrid meta-heuristics method based on genetic algorithms which combines stochastic methods to improve their performances on global optimisation just like it can be done for deterministic methods for local search, see Section 2.1.1.4; MA-LS-Chain uses three different components, integrating the CMA-ES as a line-search component,

VNS or Variable Neighbourhood Search [García-Martínez and Lozano, 2009a,b] is also a hybrid meta-heuristics method with the initial generation component being the CMA-ES.

4.4.2 Test Functions and Methods

We briefly present here the function testbeds of our benchmarking suite which are fully described in [Finck et al., 2009a,b, Hansen et al., 2009b,c]. We also describe the experimental methodology used, see also [Hansen et al., 2009a].

4.4.2.1 Test Functions

The testbeds of the noiseless and of the noisy functions [Finck et al., 2009a,a, Hansen et al., 2009b,c] respectively comprise twenty-four and thirty test functions. These functions have been chosen to display some of the typical difficulties in BBO described before in Section 4.1. We hope this collection of functions reflects a more difficult portion of the problem distribution in practice, since we consider easier problems of lesser interest. In order to make relatively simple problems less regular, transformations are applied to some of the functions in our test suite to break their symmetry or to make the functions non-linear. Rotations of the search space are considered as well to render non-separable some functions which are originally separable. And finally, there are translations of the search space —obtained by adding a fixed vector to elements of the search space— and of the function value space —by adding a fixed-value to the values of the elements of the search space.

The choice for a comprehensible function collection is motivated by the goal of analysing and understanding the behaviours of algorithms, and eventually to find ways of improving the benchmarked algorithms. The problems are split into groups: separable functions, functions with low or moderate conditioning, functions with high conditioning and uni-modal, multi-modal with adequate global structure, multi-modal with weak global structure.

All problems in our test suite are scalable with the dimension. All the functions are defined on $\mathbb{R}^n$. Each of the functions has instances: the position and value of the optimum, $\boldsymbol{x}_{\text{opt}}$ and $f_{\text{opt}} = f(\boldsymbol{x}_{\text{opt}})$ respectively, and transformations of the search space depend on the instance considered. The experimental set-up required the participants to test each function over five different instances. For statistical significance of the result of algorithms that would have a stochastic component, it was recommended that the experiment be repeated three times for each instance. This sums up to fifteen repetitions of the experiment for an algorithm on a given function. For the sake of simplicity, every time we will refer to reaching a target function value $f_{\text{target}} = f_{\text{opt}} + \Delta f$ by the precision Δf reached instead.

All functions have their optimum in the $[-5, 5]^n$ range, though most of the functions have their optimum in the $[-4, 4]^n$ range. The initialisation range was $[-5, 5]^n$ except for the IPOP-sep-CMA-ES for which it was $[-4, 4]^n$. The starting point of the algorithms was chosen uniformly in this domain. The dimensions for all functions

tested are $n = 2, 3, 5, 10, 20, 40$; 40 being optional. The success criterion of a run is defined by the fact that a target function value is reached.

4.4.2.2 Methods

We describe here the experimental methodology of BBOB 2009 [Hansen et al., 2009a]. The participants were allowed to make use of only this information: the dimensionality, the initialisation range. The target function value was provided only to prevent experiments running too long. Other than the target function value, the set of stopping criteria of the algorithm is for the participants to decide.

Performance Measure One of the performance measures that is provided in the BBOB 2009 is the ERT, *Expected Running Time*. The ERT to reach a target function value is the number of function evaluations over all runs divided by the number of runs that surpassed the considered function value. The ERT estimates the expected number of function evaluations for the target function value to be reached if the algorithm is restarted until a single success (supposing infinite time horizon). Contrarily to the SP1, no assumption is made on the number of function evaluations of an unsuccessful run. Notably, the ERT decreases if unsuccessful runs are stopped earlier. This performance measure provides quantitatively comparable measurements even for custom stopping criteria. We will refer in the following to either running times, running lengths, cost or budget without distinction from this point on. The dispersion of the ERT is determined using bootstrapping, see Appendix B.

A type of figure —an alternative to the convergence graph— that we use to present our results is the empirical cumulative distribution function of the bootstrap distribution of ERT divided by n to reach a given target function value. The empirical cumulative distribution functions of the expected running times are comparable to the data profiles and the performance profiles proposed in [Moré and Wild, 2009], it displays the empirical cumulated probability of success on the problems considered depending on the allocated budget. The probability of success can be considered over all functions of a testbed over multiple target function values.

We provide more details on the empirical cumulative distribution functions of the bootstrap distribution of ERT divided by n in Appendix B.

Timing Experiments We required that the participants provide results of their algorithm on the following time complexity experiment: the overall CPU time of the execution of the algorithm software optimising function f_8 is measured. This measurement is required to last at least a few tens of seconds and at least a few iterations of the algorithm. If any stopping criterion is reached, the algorithm is to be restarted. This experiment is made to reflect a standard execution of the algorithm. The CPU time per function evaluation is reported for each dimension. Again a software implementation of such experiment is provided.

Crafting Effort An additional measure of an algorithm facing the BBOB 2009 is the crafting effort which evaluates the versatility of the algorithm [Feoktistov, 2006, Price, 1997]. The crafting effort is expressed as follows: $\mathrm{CrE} = -\sum_{k=1}^{K} \frac{n_k}{n} \ln(\frac{n_k}{n})$, where K is the number of different parameter settings used over a whole BBOB 2009 testbed, n is the number of functions in the considered testbed, the n_k are the number of functions which used the same k-th setting. The crafting effort is positive, equal to zero if a single setting was used. As a matter of fact, the crafting effort is zero for all algorithms that we have benchmarked.

Algorithms We thereafter describe the experimental set-up for the algorithms presented previously in Section 4.4.1.

For both the BFGS method and NEWUOA, a maximum of a hundred restarts is used.

For the BFGS method, the same MATLAB implementation we already tested is used, see Section 4.3. The stopping criteria are set so a restart of the algorithm happens only when line search fails due to round-off errors. A maximum of 10^5 function evaluations times the dimension of the search space is used, the algorithm stops in case of success or when the number of function evaluations or the number of restarts runs out, see Listing 4.1.

For the NEWUOA as well, the implementation used for our experiments is the one we tested previously in Section 4.3. We consider for q the number of interpolation points the values $2n + 1$ (recommended by Powell [2006]) the rounded value of $\sqrt{(n + 1/2)(n + 1)(n + 2)}$ (average model) and the maximum $\frac{(n+1)(n+2)}{2}$ (full model) where n is the dimensionality of the problem. These variants of the NEWUOA will

Listing 4.1: MATLAB code: Multi-start BFGS

```
function [x, ilaunch] = MY_OPTIMIZER(FUN, D, ...
                        ftarget, maxfunevals)
% minimizes FUN in D dimensions by independent restarts
% of fminunc (BFGS). ftarget and maxfunevals are
% additional external termination conditions.
% Search space is [-5, 5]^D

  % set options, make sure we always terminate
  options = optimset('fminunc');
  % BFGS algorithm
  options = optimset(options, 'LargeScale', 'off');
  options = optimset(options, 'MaxIter', inf, ...
                  'Tolfun', 1e-11, 'TolX', 0, ...
                  'OutputFcn', @callback, ...
                  'Display', 'off');

  maxfunevals = min(1e4*DIM, maxfunevals);

  % multistart such that ftarget is reached with
  % reasonable prob. Relaunch optimizer up to 100 times
  for ilaunch = 1:100;
    options = optimset(options, 'MaxFunEvals', ...
              maxfunevals - feval(FUN, 'evaluations'));
    x = fminunc(FUN, 10*rand(DIM,1)-5, options);
    if (feval(FUN, 'fbest') < ftarget || ...
        feval(FUN, 'evaluations') >= maxfunevals)
      break;
    end
  end

  function stop = callback(x, optimValues, state)
    stop = false;
    if optimValues.fval < ftarget
      stop = true;
    end
  end
end % function
```

be subsequently denoted as NEWUOA, NEWUOA (avg) and NEWUOA (full) respectively. The initial radius of the trust region was set to 10. The stopping criterion which is the final radius of the trust region is set to 10^{-16}. The maximum number of function evaluations is 10^5 for NEWUOA, 10^4 for the NEWUOA (full), 10^5 and 10^4 for NEWUOA (avg) on the noiseless and noisy testbeds respectively.

For the IPOP-sep-CMA-ES, the MATLAB implementation of the CMA-ES (version 3.23beta) is used[10]. The sep-CMA-ES variant is used only for the first $1 + 100n/\sqrt{\lambda}$ iterations of the *first start*. The initial step-size was set to 2. All other parameters are set to their default values. Restarts occur after $100 + 300n\sqrt{n/\lambda}$ iterations or if any of the default stopping criterion is met. A maximum of 8 restarts or 10^4 function evaluations is used.

The Monte Carlo search was implemented in both MATLAB[11] and C, the latter essentially for a reason of time performance. The maximum number of function evaluations was set to 10^5.

4.4.3 Results

We present the results of the algorithms considered on the timing experiments and testbeds presented in the following.

4.4.3.1 Timing Experiment

All experiments were done on a Intel Core 2 6700 processor (2.66 GHz) on Linux 2.6.24.7. Results can be found in Table 4.2. The CPU time of a function evaluation of the Monte Carlo search grows linearly with the dimension. For the NEWUOA, the CPU time increases with the dimension, scaling worse as the model grows more complex.

For the BFGS, the decreasing CPU time may be due to initialising process mainly representing the cost of the function evaluation. We assume the CPU time per function evaluation would increase given that the dimensionality is large enough.

For the variants of the CMA-ES with increasing population (IPOP), two behaviours can be identified. The behaviour of the IPOP-sep-CMA-ES and the IPOP-CMA-ES shows that up to 10-D, the necessary CPU time decreases with increasing

[10] Latest version available here: http://www.lri.fr/~hansen/cmaesintro.html
[11] An example implementation is provided in Listing 5.1

Table 4.2: CPU time per function evaluation in microseconds, next are two figures between brackets: the corresponding number of runs, and the average number of restarts. IPOP-always-sep stands for an increasing population size space and time linear variant sep-CMA-ES, whereas IPOP-sep stands for the IPOP-sep-CMA-ES, which only uses sep-CMA-ES for a few initial iterations.

	2-D	3-D	5-D	10-D
BFGS	600	470	370	300
NEWUOA	8.1	11	21	58
NEWUOA (avg)	8.0	13	27	100
NEWUOA (full)	9.0	15	38	240
IPOP-CMA-ES	170 (271/0)	140 (180/0)	120 (88/0)	100 (45/0)
IPOP-sep	150 (194/0)	130 (141/0)	110 (81/0.2)	95 (39/>0)
IPOP-always-sep	18 (18/ 7)	24 (13/6)	38 (8/5)	65 (5/2)
Random Search	0.12	0.16	0.23	0.45

	20-D	40-D	80-D
BFGS	290	290	280
NEWUOA	170	620	2500
NEWUOA (avg)	580	3900	-
NEWUOA (full)	2400	32000	-
IPOP-CMA-ES	100 (13/0)	130 (5/0.2)	310 (5/0)
IPOP-sep	96 (8/0.5)	120 (5/0)	220 (5/0)
IPOP-always-sep	68 (5/1)	56 (6/0)	53 (6/0)
Random Search	0.88	1.7	3.4

dimension, presumably due to a larger number of initialisation procedures for the required multiple runs of the algorithms until thirty seconds have passed. Otherwise a quadratic scaling of the internal time complexity is expected for both algorithms with a slightly better scaling for the IPOP-sep-CMA-ES. The behaviour for the IPOP-always-sep-CMA-ES shows that the CPU-time increases for dimension up to 20-D before decreasing. This is due to both the number of internal restarts of the algorithm and to the implementation of MATLAB. The numerous and increasing restarts of the algorithm when we consider dimensions from 20 to 2 result in having more iterations of the algorithm with a larger population size. Our implementation in MATLAB results in a a smaller CPU time per function evaluation for a larger population size. We assume that the observed decrease of the CPU time per function evaluation in 40-D and 80-D successively must be followed by an increase in larger dimension as for the IPOP-sep-CMA-ES or IPOP-CMA-ES.

The results for the variants of CMA-ES with increasing population size (IPOP) show dependencies with the number of restarts: due to MATLAB implementation the larger the population size, the faster the algorithm is.

4.4.3.2 Performance Results of NEWUOA, IPOP-sep-CMA-ES, BFGS, Monte Carlo search

In addition to the empirical cumulative distribution functions of the running times that are presented here and which show results of the algorithms on multiple functions altogether, more detailed results can be found in [Auger and Ros, 2009a,b, Ros, 2009a,b,c,d,e,f].

The results of the Monte Carlo search in terms of ERT, for instance in Figure 4.11, show that to reach the the target function values 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7} is gradually harder as the dimensionality of the problem increases. Also the targets are not to be uniformly distributed over the different objective functions considered: an arbitrary precision on a given problem, say $\Delta f = 10^{-8}$ on the Gallagher function f_{22} in 2-D can be attained, in this case in around a million function evaluations on fifteen out of fifteen trials, whereas the same precision can never be achieved on another problem, for instance for the same target function value on the sphere function f_1 in 2-D.

Comparison results of the different algorithms can be seen in Figures 4.12 to

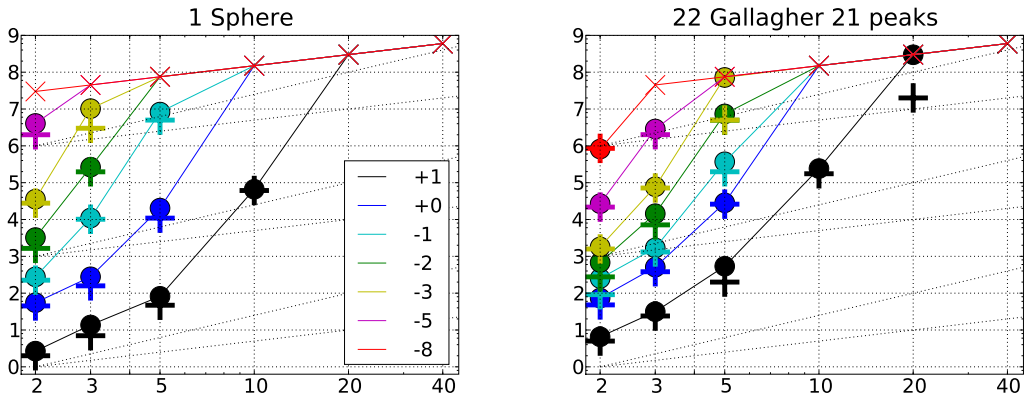


Figure 4.11: Expected Running Time (ERT, ●) and number of function evaluations of the median trial (+) for the Monte Carlo search to reach the target function value $10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ versus dimension in log-log presentation. The ERT equals to the number of function evaluations to reach the target function value divided by the number of successful trials, where a trial is successful if the target function value was surpassed during the trial. Crosses (×) indicate the total number of function evaluations. Annotated numbers on the ordinate are decimal logarithms. Additional dashed lines show linear and quadratic scaling.

4.16 in 2 to 20-D respectively. Though the Monte Carlo search converges to the optimum given enough time, the dimensionality greatly affects the performances of the Monte Carlo search, reducing its probability of success from around seventy and eighty percents in 2-D on the noiseless and noisy testbeds respectively to around five and ten percents in 20-D, over all the target function values considered over all functions.

For the noiseless testbed (top sub-figures of Figures 4.12 to 4.16) NEWUOA fares best for targets reachable within less than a few hundreds function evaluations times the dimension. Then, IPOP-sep-CMA-ES surpasses all algorithms for targets reachable within more than one thousand function evaluations times dimension. On the noiseless testbed, the main difference between NEWUOA, IPOP-sep-CMA-ES and BFGS is on the multi-modal functions, see Figure 4.17, as opposed to uni-modal functions, see Figure 4.18, where BFGS and NEWUOA do not reach a target function value smaller than 10^{-4} for a dimension larger than 5-D, barely performing better than the Monte Carlo search. Otherwise, except for certain intermediate range of

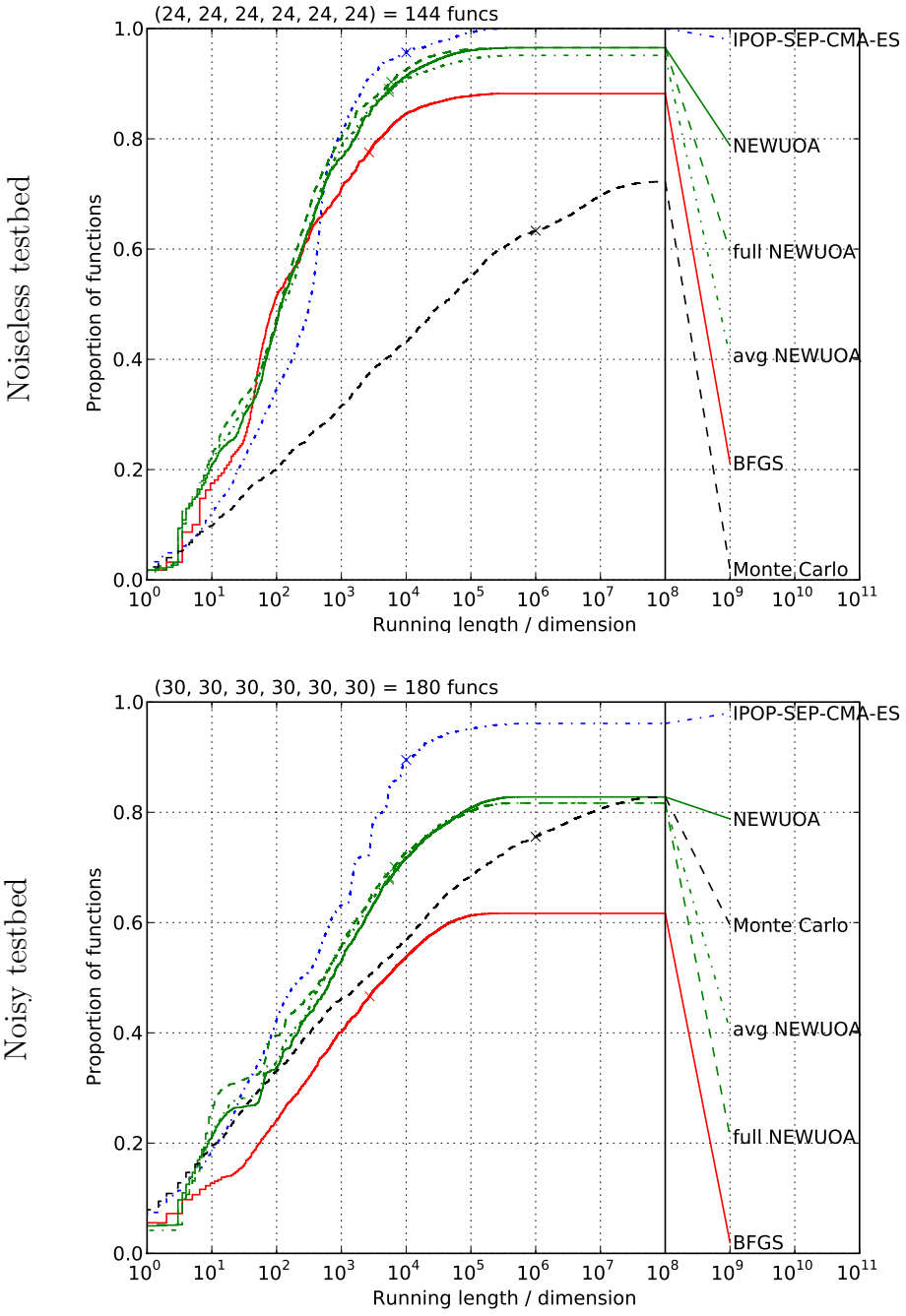


Figure 4.12: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 2-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

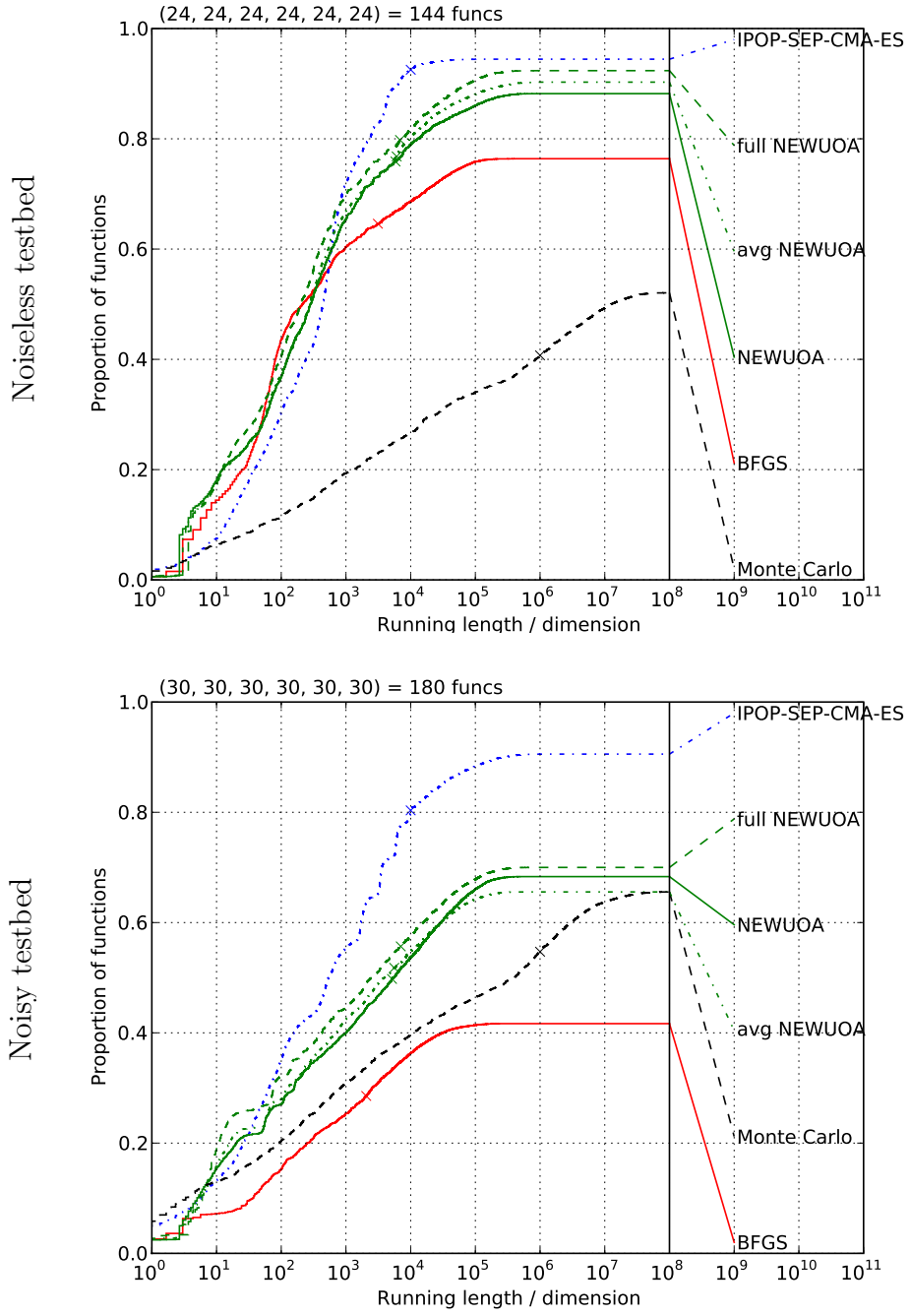


Figure 4.13: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 3-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

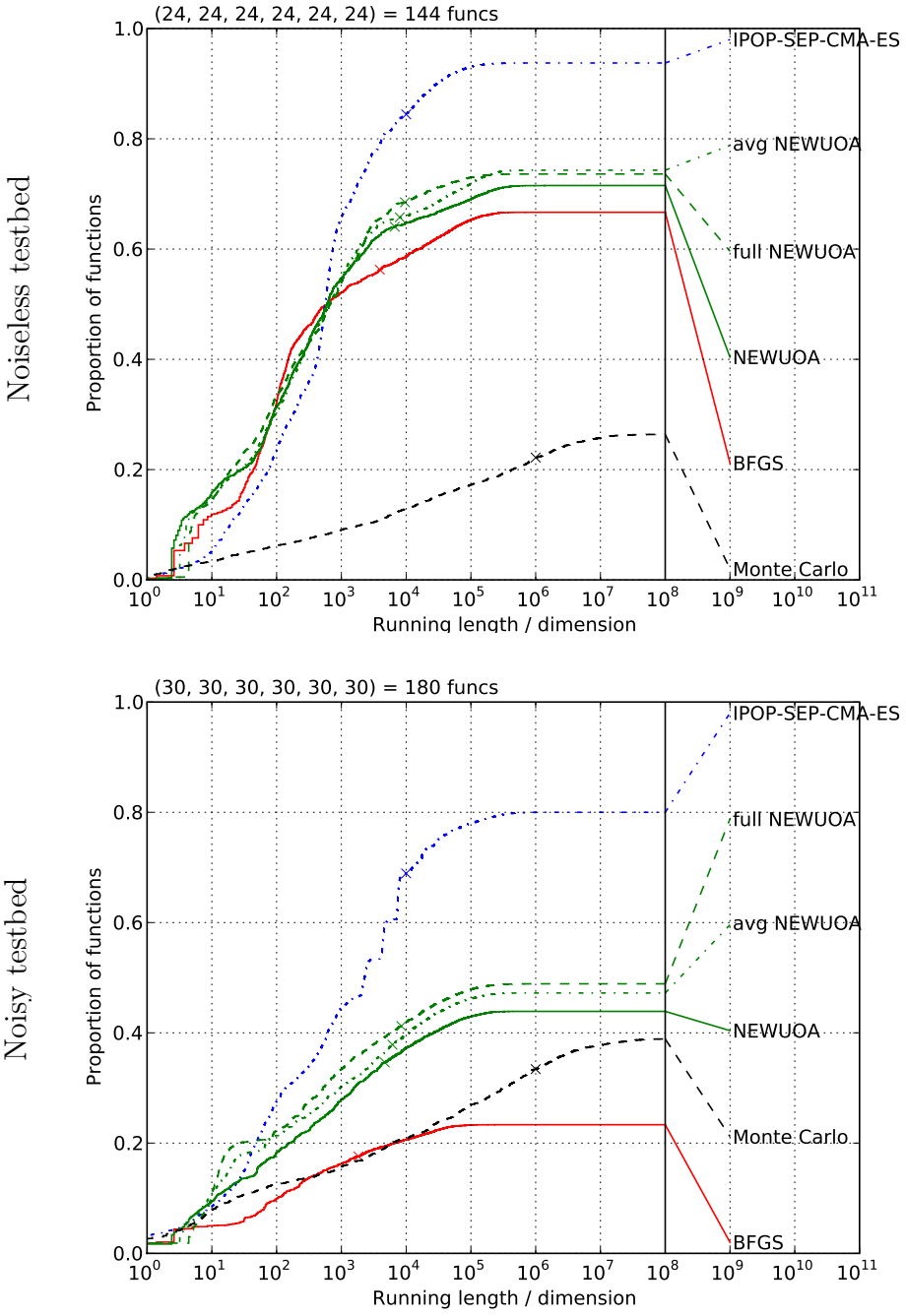


Figure 4.14: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 5-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

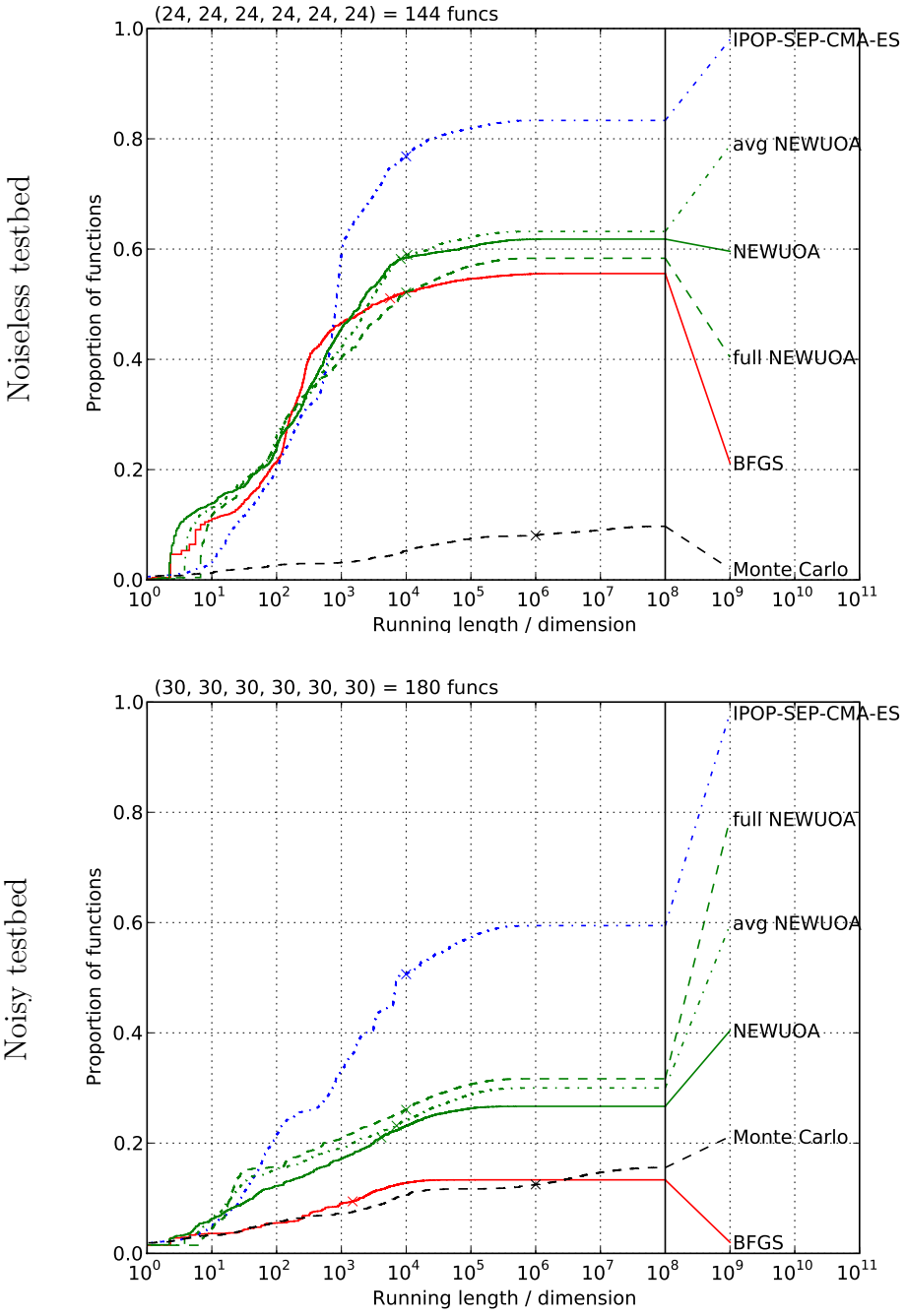


Figure 4.15: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 10-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the maximum number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

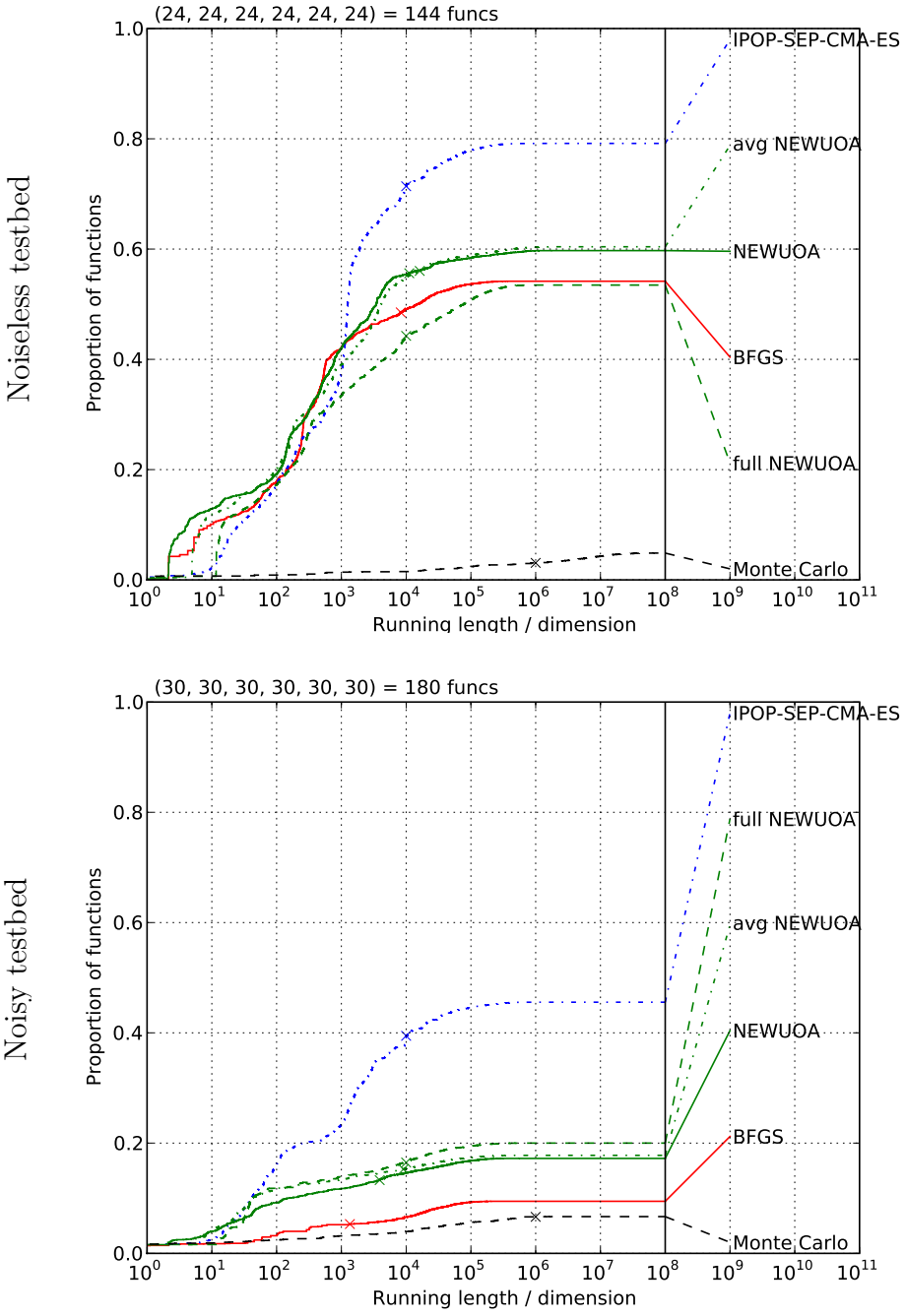


Figure 4.16: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 20-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the maximum number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

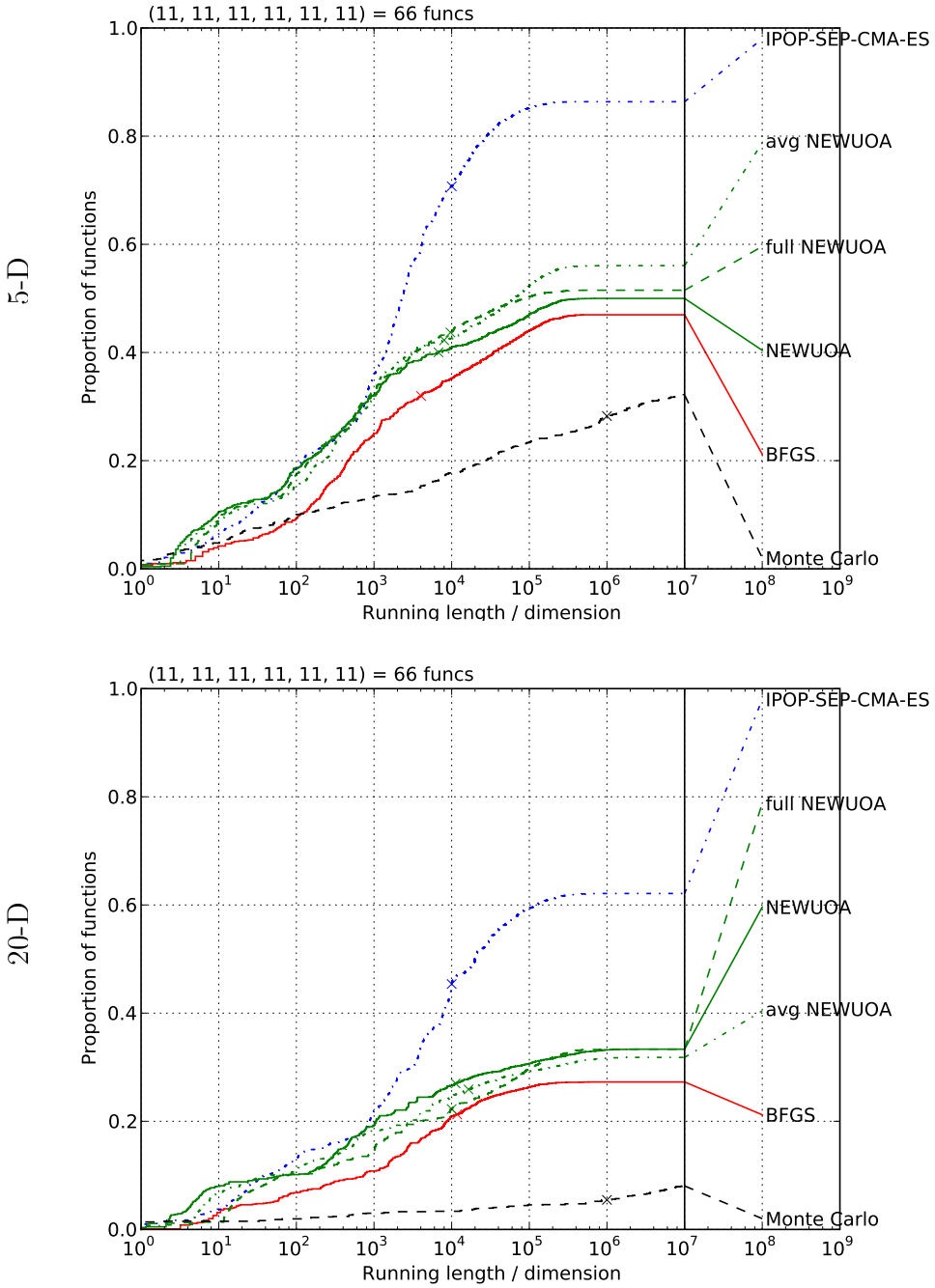


Figure 4.17: Empirical cumulative distribution of the running lengths divided by dimension to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7} on the *multi-modal* functions f_4, f_{15} to f_{24}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

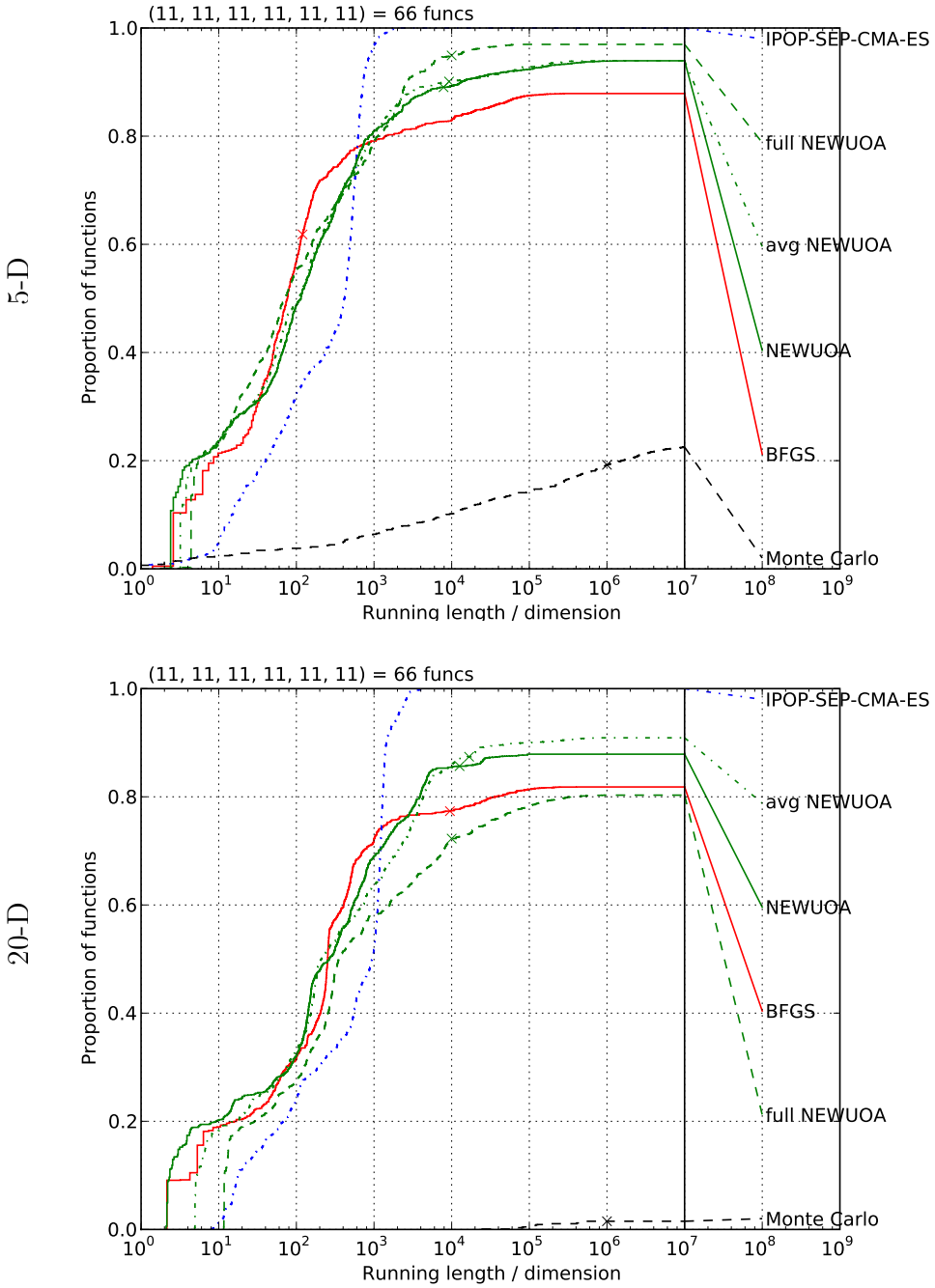


Figure 4.18: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7} on the *uni-modal* functions f_1, f_5 to f_{14}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

budgets in lower dimensions, the NEWUOA is performing better than BFGS on the noiseless testbed.

For the noisy testbed (bottom sub-figures of Figures 4.12 to 4.16), the IPOP-sep-CMA-ES performs the best over all targets, only followed by the NEWUOA (full). This may be explained by the stochastic behaviour of the IPOP-sep-CMA-ES. Of the variants of NEWUOA, the NEWUOA (full) performs best because of the larger number of interpolation points for the model.

4.4.3.3 Performance Results of all BBOB 2009 entries

We present here the comparison results of all the algorithms listed in Section 4.4.1.3. We skip the discussion on the timing experiment results, and specify here that all algorithms presented had a crafting effort of zero, except for GLOBAL which featured in 10 and 20-D a crafting effort of 0.51 and 0.66 on the noiseless and noisy testbeds respectively —using another different internal local search method for five out of the twenty-four noiseless functions, and eleven out of the thirty noisy functions. The performances are expressed in term of expected running time, ERT, for an algorithm to surpass a target function value.

Considering Figures 4.19 to 4.23 which corresponds to functions from 2 to 20-D, the dimensionality affects the performances of all algorithms to a greater or lesser extent depending on the algorithm. This affects the ranking of the algorithms over the dimensionality. On the noiseless testbed, in 2 and 3-D, the success probability of the algorithms are quite comparable and range from a hundred to seventy percent, except for the Monte Carlo search and BayEDA$_{\mathrm{cG}}$. Only in dimensions larger than 5-D do the performances of the algorithms spread out. The Nelder-Mead performs well in 2 and 3-D, but is outperformed in larger dimensions.

The ranking of algorithms also depends on the running length or budget considered. Some algorithms such as POEMS and LSstep only perform better than the Monte Carlo search for budgets larger than two hundred function evaluations times dimension on the noiseless testbed.

Such an extended number of submissions gives a global view over the different techniques available to the practitioner and allows to compare the performances of algorithms ranging from DASA (Ant-Colony based) to the Rosenbrock's local search

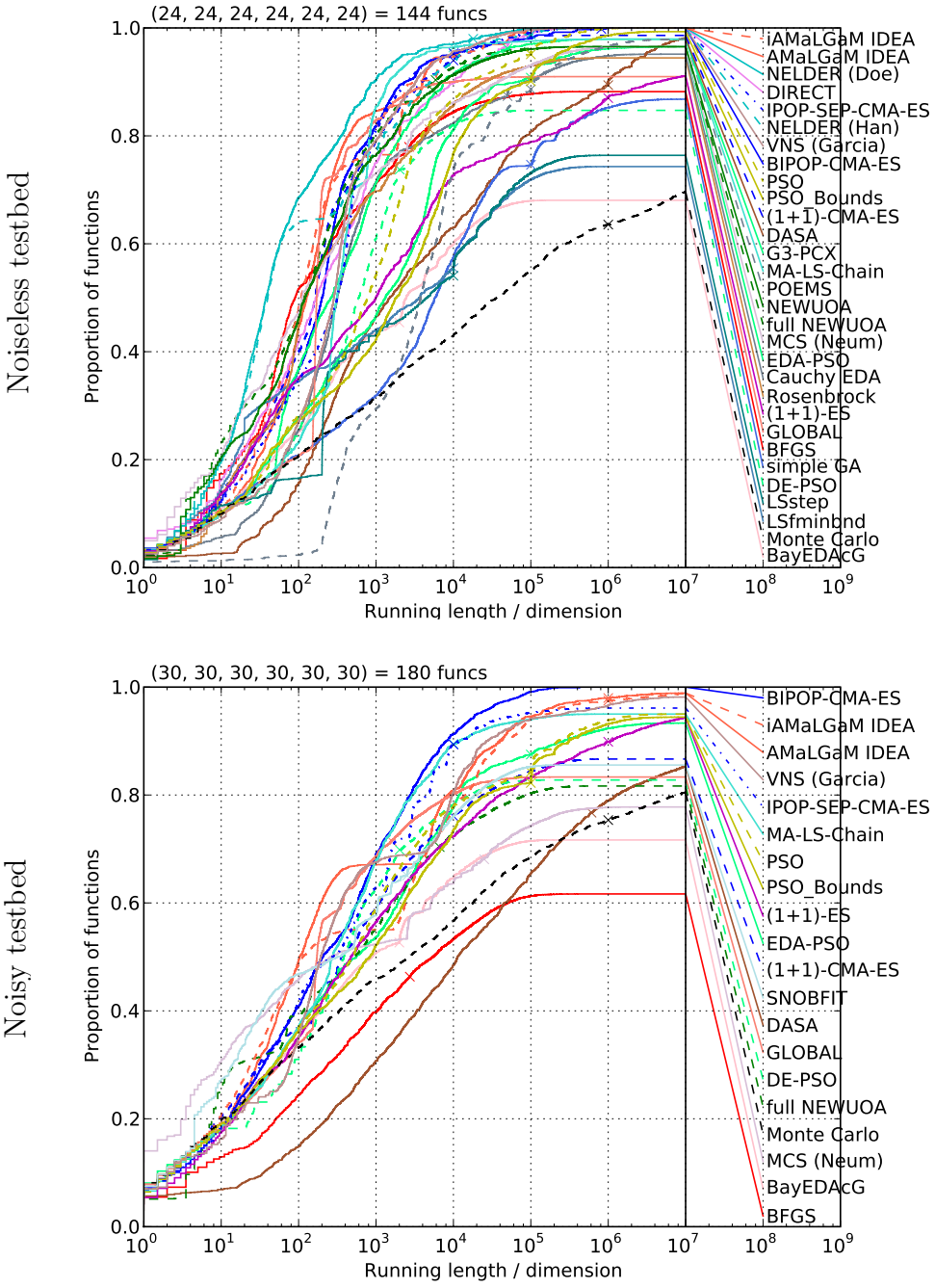


Figure 4.19: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 2-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

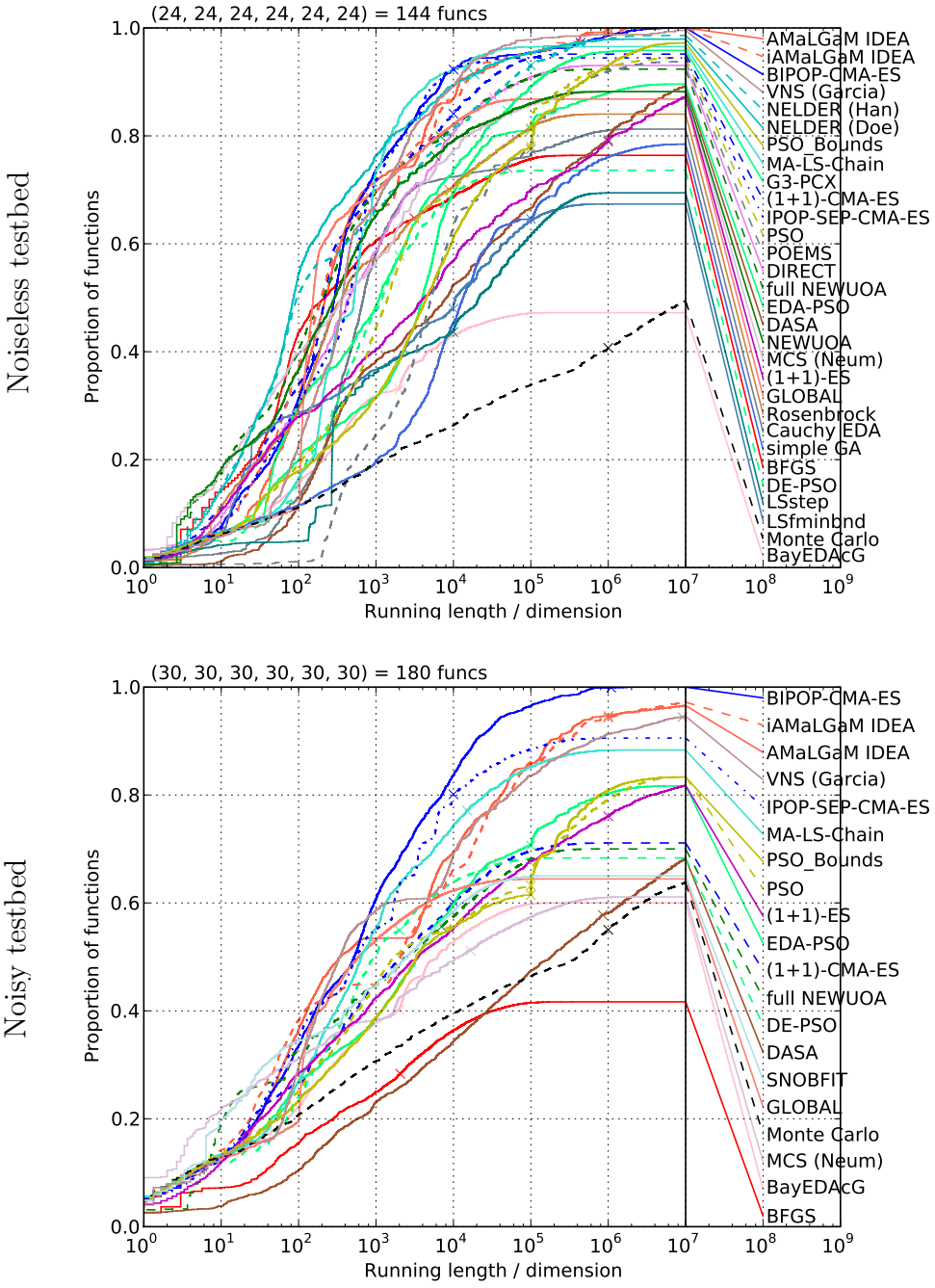


Figure 4.20: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 3-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

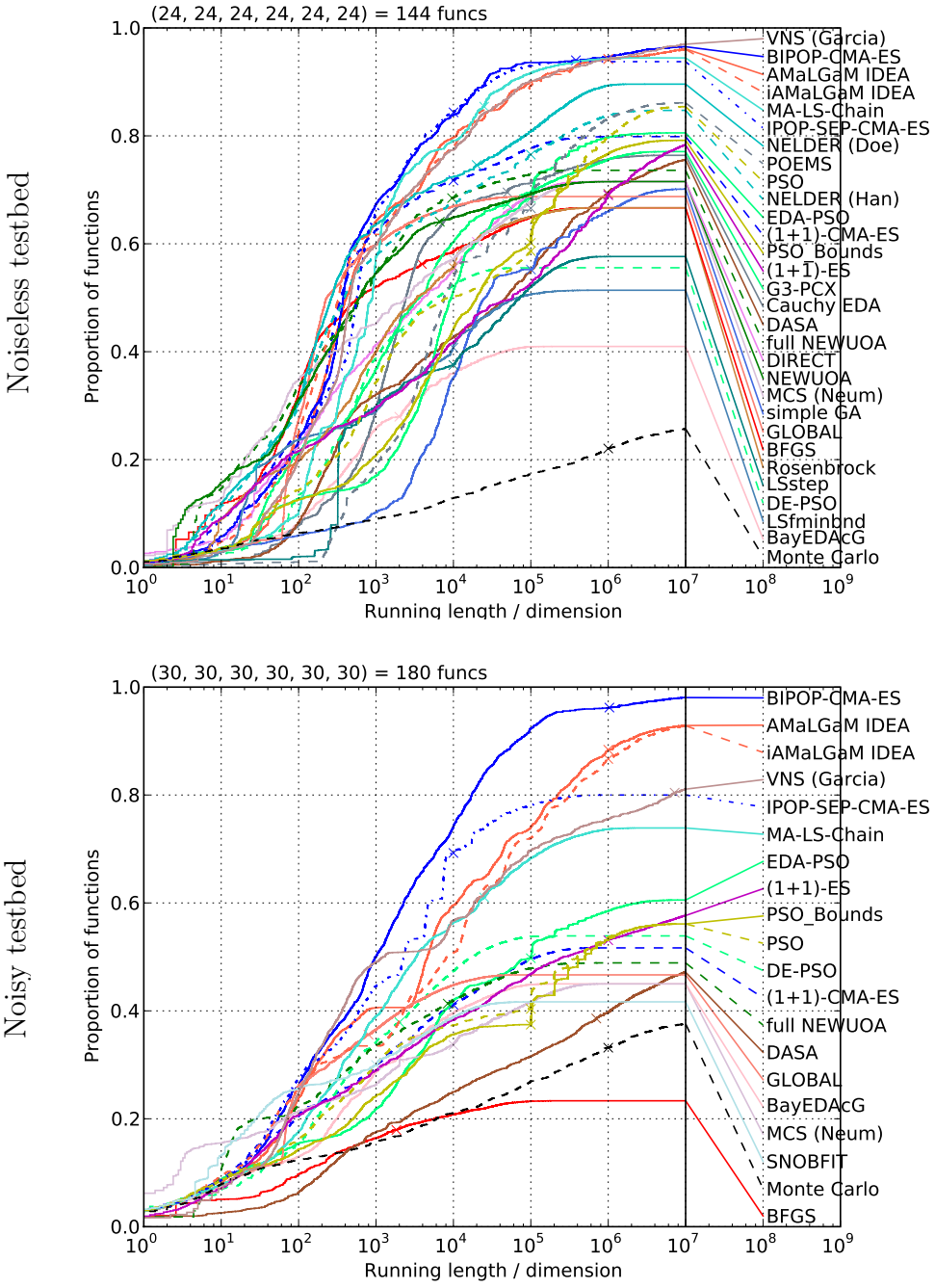


Figure 4.21: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 5-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

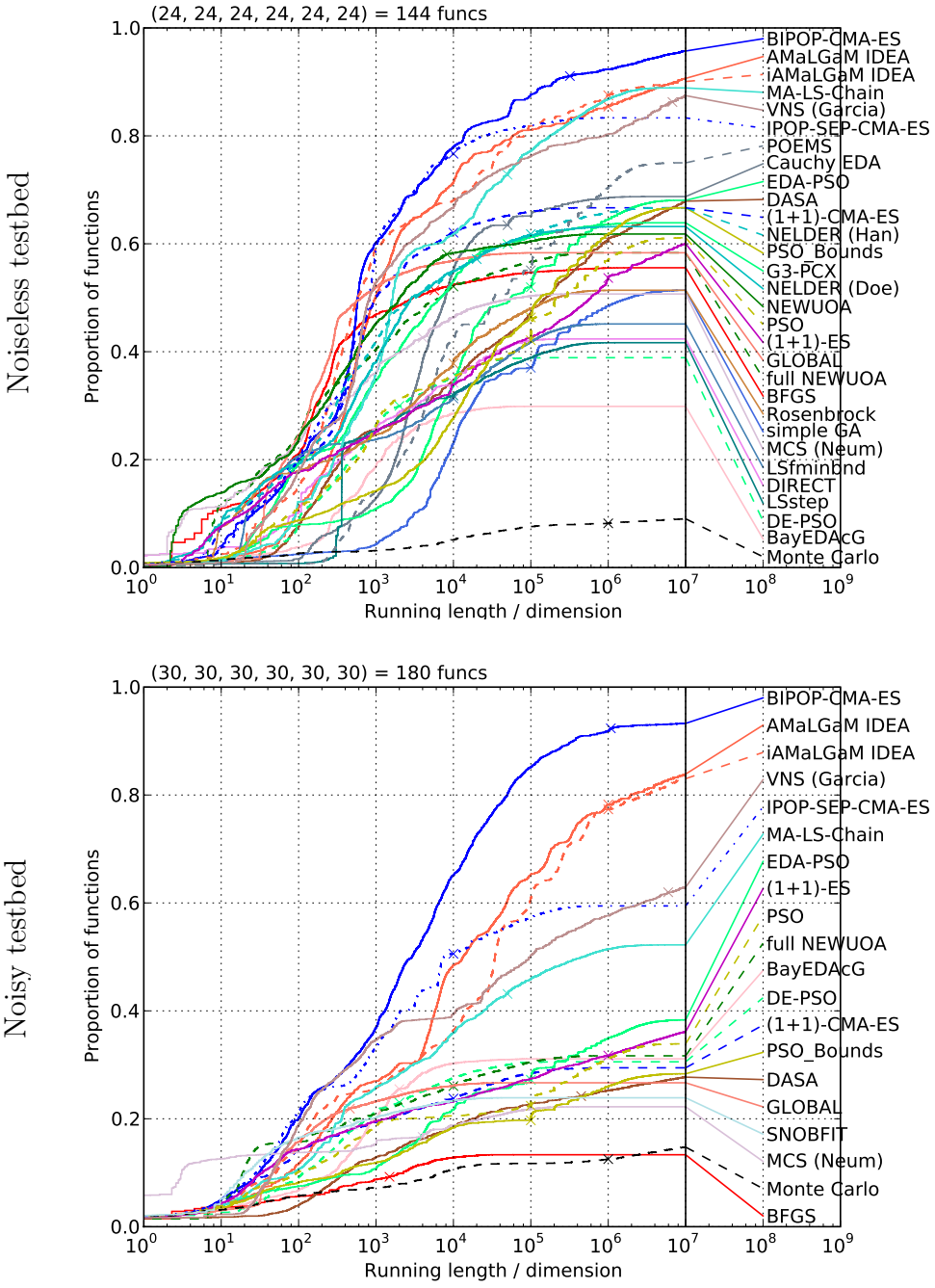


Figure 4.22: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 10-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

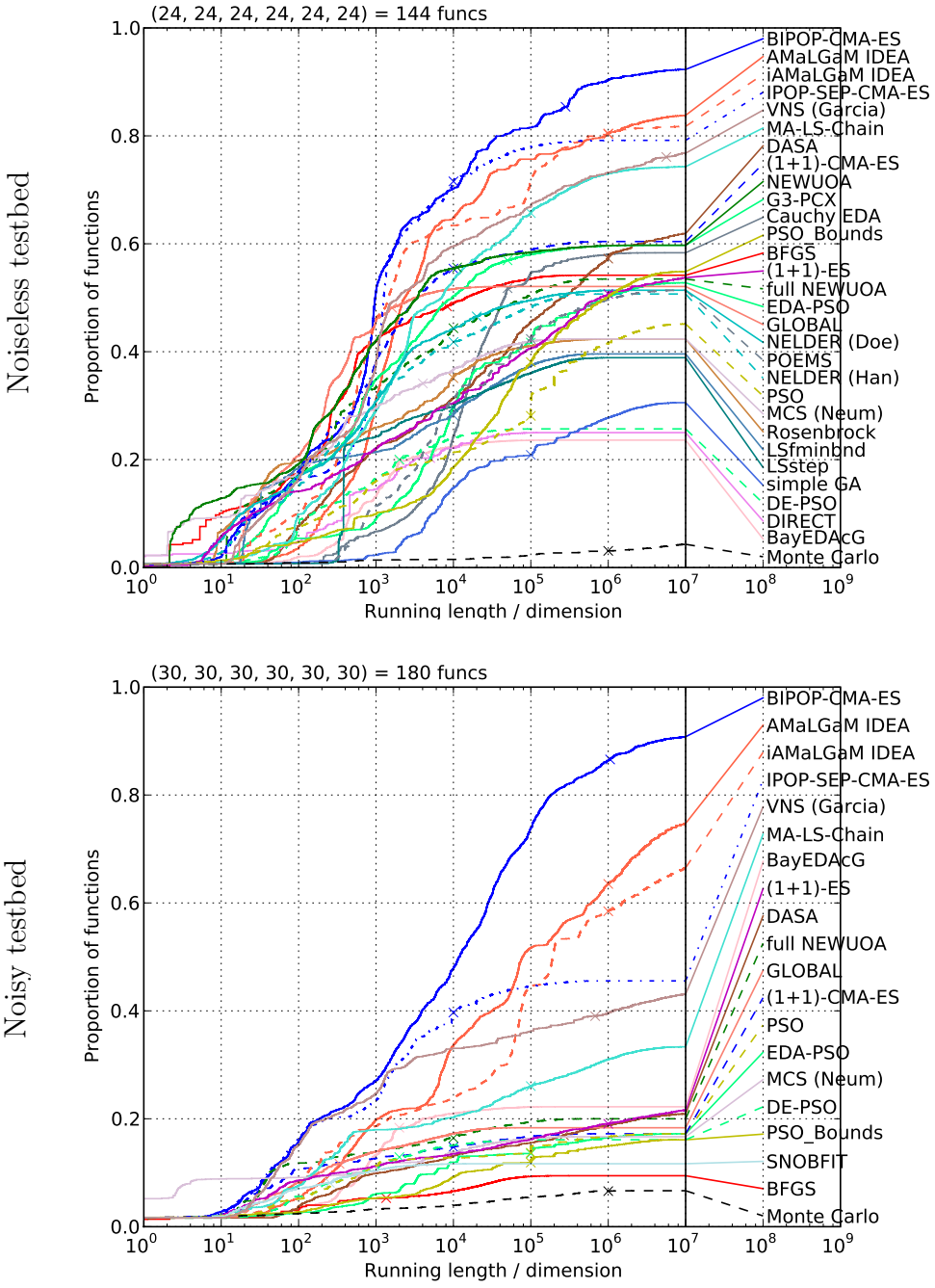


Figure 4.23: Empirical cumulative distribution function of the bootstrap distribution of the running lengths divided by dimension (here 20-D) to reach target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7}. The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

method, the LSstep and novel algorithms like the NEWUOA. Compared to the Monte Carlo search, the (1+1)-ES, for instance, performs consistently better over the two testbeds in all dimensionalities whereas some methods fail to overcome specific difficulties encountered in optimisation for instance noise in the case of the BFGS algorithm.

Overall, the MCS algorithm shows interesting results for budgets smaller than ten times dimension function evaluations. For even one function evaluations, its probability of success is significantly better than that of other algorithms especially on the noisy testbed. An explanation is the initialisation of the MCS algorithm where the origin of the search space is evaluated.

On the noiseless testbed (top sub-figures of Figures 4.19 to 4.23) and Figure 4.24, NEWUOA and BFGS both perform well for budgets smaller than a hundred function evaluations. On the noiseless testbed ranging from 5 to 20-D, the GLOBAL algorithm performs the best for an intermediate budget between a hundred and a thousand function evaluations.

Given that the dimension of the search space and the budget are large enough the BIPOP-CMA-ES outperforms all other algorithms. The AMaLGaM IDEA, iAMaLGaM IDEA, MA-LS-Chain, VNS and IPOP-sep-CMA-ES all perform relatively close to BIPOP-CMA-ES.

The main differences between the BIPOP-CMA-ES and IPOP-sep-CMA-ES are: 1. the ability of the BIPOP-CMA-ES to maintain small population capabilities after a number of restarts which allow the BIPOP-CMA-ES to perform better than IPOP-sep-CMA-ES on the Lunacek bi-Rastrigin function f_{24}, see Figure 4.24, 2. their maximum number of function evaluations, as can be seen in Figures 4.22 and 4.23 for instance. These differences explain the difference in the probability of success between the two algorithms.

On the noisy testbed (bottom sub-figures of Figures 4.19 to 4.23), the BIPOP-CMA-ES is outperforming all algorithms in all dimensions when the budget is larger than around two thousand function evaluations times dimension in 2-D, around a hundred function evaluations times dimension for 10 and 20-D.

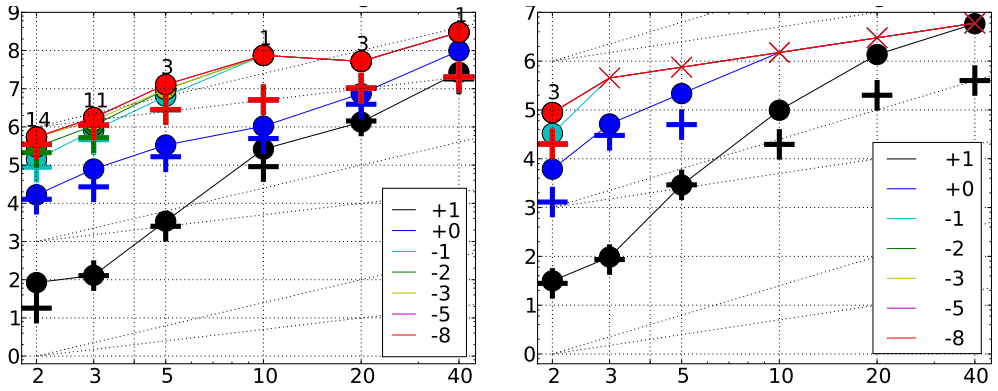


Figure 4.24: On the function f_{24} of BBOB, Expected Running Time (ERT, ●) and number of function evaluations of the median trial (+) for BIPOP-CMA-ES (left) and IPOP-sep-CMA-ES (right) to reach the target function value $10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ versus dimension in log-log presentation. The ERT equals to the number of function evaluations to reach the target function value divided by the number of successful trials, where a trial is successful if the target function value was surpassed during the trial. Crosses (×) indicate the total number of function evaluations. Annotated numbers on the ordinate are decimal logarithms. Additional dashed lines show linear and quadratic scaling.

Uni-modal versus Multi-modal functions, The probability of success of the algorithms tested is higher on uni-modal functions than on multi-modal functions overall, see Figure 4.25. Furthermore we can see algorithms demonstrate different behaviour on uni-modal and multi-modal functions. Especially BFGS and NEWUOA perform among the bests on uni-modal functions for budgets smaller than a thousand times n function evaluations. On multi-modal functions though, the performances of BFGS are not so remarkable, whereas the performances of NEWUOA are still among the bests for budgets smaller than a thousand times the dimension function evaluations. When the budget is larger, BIPOP-CMA-ES AMaLGaM IDEA, iAMaLGaM IDEA, and IPOP-sep-CMA-ES outperform all other algorithms.

Figure 4.26 shows the performances of all algorithms on the different groups of functions in the noiseless testbed. The line search methods LSstep and LSfminbnd perform rather well on the separable functions, and so do the PSO (PSO_Bounds especially), POEMS and DASA. Conversely, BIPOP-CMA-ES, IPOP-sep-CMA-ES, AMaLGaM IDEA, iAMaLGaM IDEA do not perform as well, mainly because of

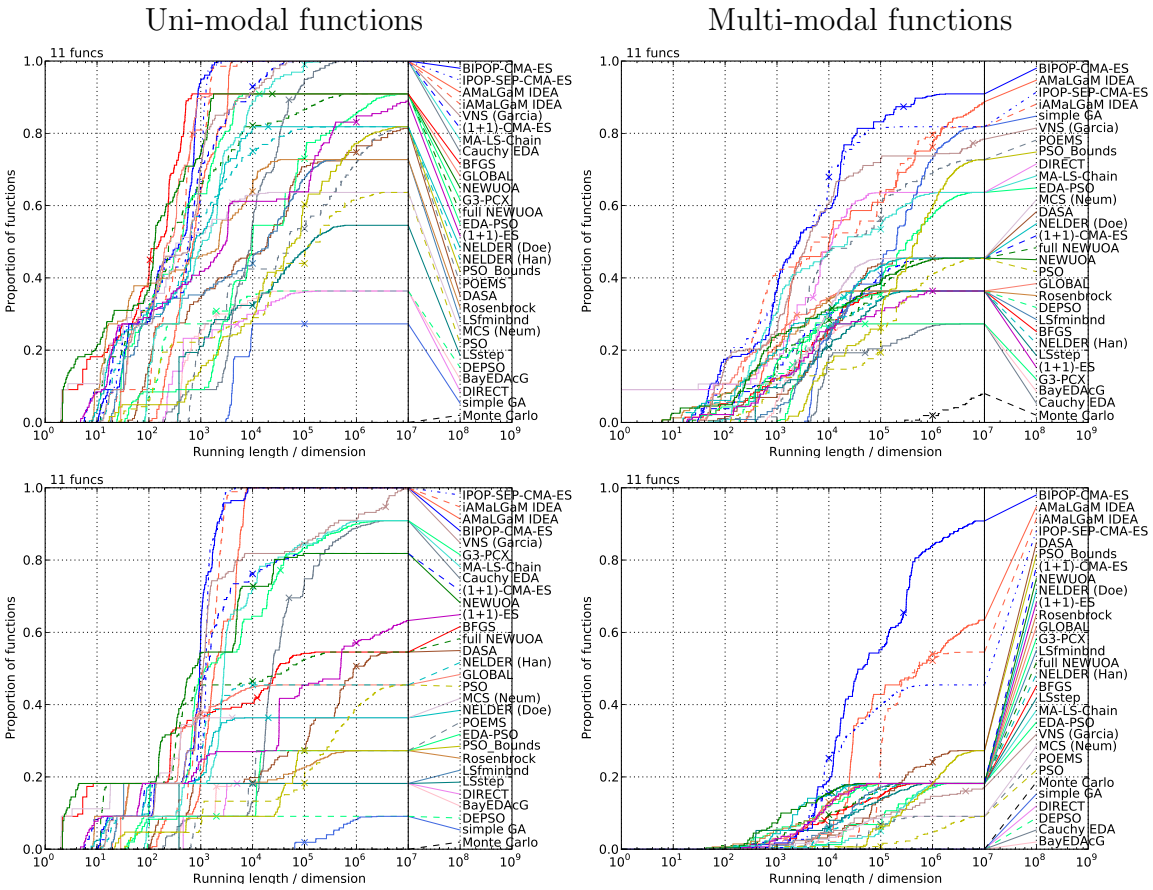


Figure 4.25: Empirical cumulative distribution function of the bootstrap distribution functions of the running lengths in 20-D for target function values of 1 (top) and 10^{-7} (bottom row) on the uni-modal functions f_1 and from f_5 to f_{14} (left) and multi-modal functions f_4 and from f_{15} to f_{24} (right column) of the noiseless testbed

their failure on the f_3 and f_4 Rastrigin functions in 20-D for the smallest target function values. The NEWUOA performs the best on functions with low or moderate conditioning when the budget is less than a thousand times n function evaluations. The NEWUOA, BFGS and GLOBAL perform the best on functions with high conditioning and uni-modal for a budget smaller than a thousand times n. For the functions with low, moderate or high conditioning, for larger budgets, BIPOP-CMA-ES, IPOP-sep-CMA-ES, AMaLGaM IDEA, iAMaLGaM IDEA perform best. These same algorithms perform best overall on the multi-modal functions with adequate global structures and on non-smooth functions, whereas on the multi-modal functions with weak global structure only the BIPOP-CMA-ES perform remarkably well compared to all the other algorithms.

4.4.4 Summary and Discussion of the Results of BBOB 2009

We have defined for the GECCO 2009 workshop session called BBOB 2009 an experimental benchmarking framework for the comparison of BBO algorithms. The results of BBOB 2009 featuring many different algorithms on a testbed of noiseless functions and another of noisy functions in dimension 2, 3, 5, 10, 20 —40-D was optional and not presented in this thesis— have been compiled and are presented here.

To assess the performances of algorithms we used the Expected Running Time, ERT, which is the total number of function evaluations for the considered algorithm divided by the number of runs that surpassed a given target function value. The ERT allowed us to aggregate the information of multiple optimisation runs and giving elements of the bootstrap distribution of ERT provided with a dispersion measure. Also, time measurements of the algorithms on the function f_8 in different dimensions were required.

We have defined the crafting effort CrE to evaluate the versatility of algorithms: the CrE computes a single positive value on one testbed that grows larger depending on the number of different settings used for the considered algorithm.

We have run the following algorithms: 1. a line search method using the BFGS update equation with independent restarts, 2. the NEWUOA also with independent

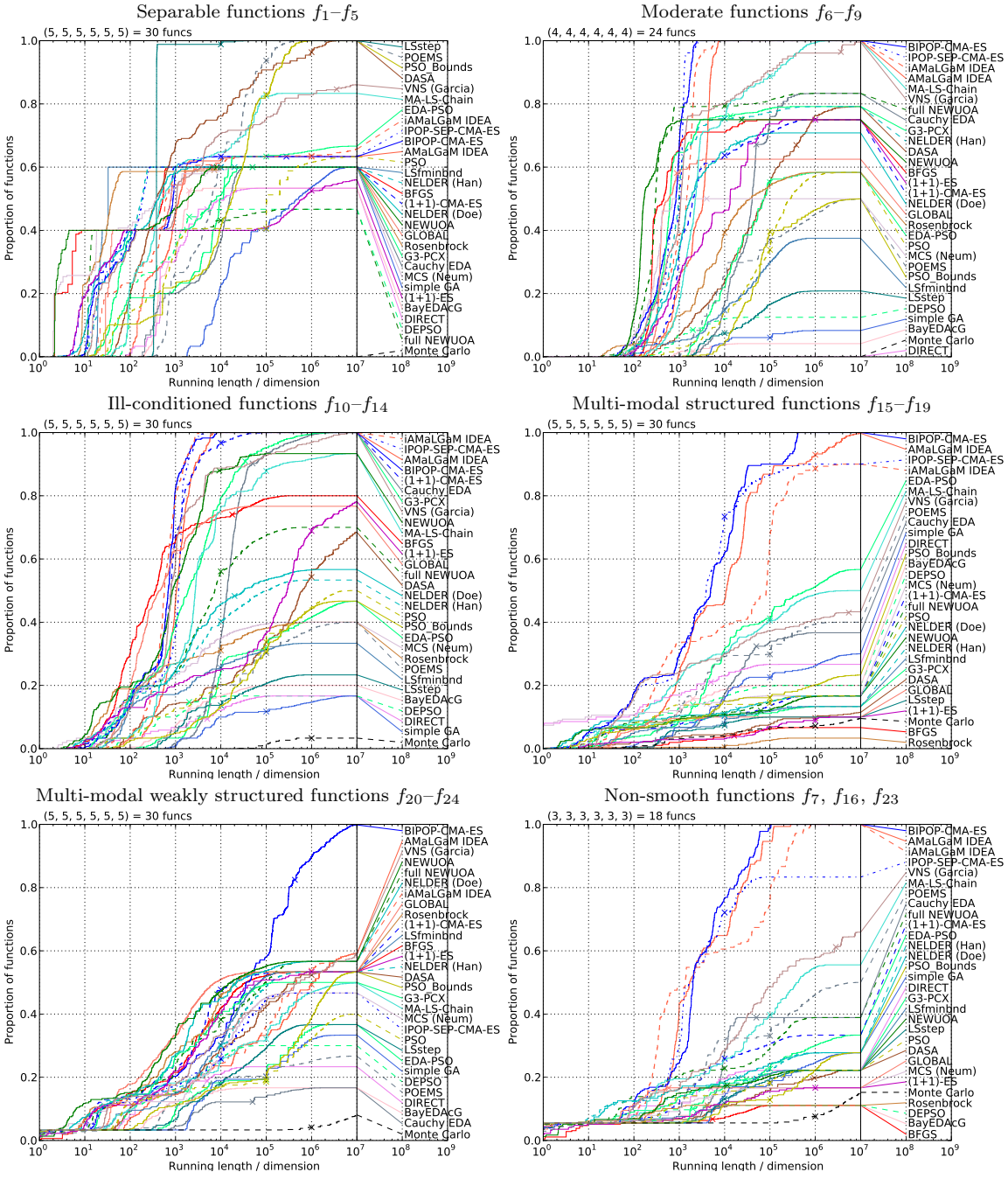


Figure 4.26: Empirical cumulative distribution function of the bootstrap distribution functions of the success probability for a given running length in 20-D for target function values of 10, 1, 0.1, 10^{-3}, 10^{-5}, 10^{-7} on different function groups of the noiseless testbed

restarts, 3. the IPOP-sep-CMA-ES, which is the switch policy sep-CMA-ES/ CMA-ES discussed in Chapter 3 with increasing population size restarts, and 4. the Monte Carlo search which samples the search space uniformly. The parameter tuning of all algorithms resulted in a crafting effort of zero for these algorithms.

The results of the Monte Carlo search are affected by the 'curse of dimensionality' since the fixed target function values 10, 1, 10^{-1}, 10^{-3}, 10^{-5} and 10^{-7} are gradually harder to reach. Furthermore, the results of the Monte Carlo search also show that reaching these target function values —which were the same for all functions in all dimensions— is not equally difficult to Monte Carlo search. Therefore, the choice of these function values, which were used to present our results, can be discussed. An alternative is to choose the function values depending on the performances of a reference algorithm as discussed briefly in Appendix B.

On the noiseless testbed, the best algorithm out of the four that we have run depends on the budget of function evaluations to reach a target function value. If the budget is smaller than a hundred times the dimension, NEWUOA fares best. If the budget is larger than one thousand times the dimension, then it is the IPOP-sep-CMA-ES that has the best performances. The BFGS performs best for an intermediate budget range. The Monte Carlo search always comes out last. On the noisy testbed, the IPOP-sep-CMA-ES is always best, followed by NEWUOA. The BFGS and Monte Carlo search perform comparably.

Many other algorithms were submitted to BBOB 2009. We have shown the results of the comparison of all of the entries of BBOB 2009 and GLOBAL, SNOBFIT and MCS which are late additions. Of all entries, only GLOBAL had a crafting effort larger than zero since GLOBAL used two different settings on the noiseless testbed as well as on the noisy testbed.

When comparing all algorithms together, considering results from 2-D to 20-D provided some interesting scaling results. In 2 and 3-D, the Nelder-Mead method is virtually the best method. In this case though, the performances of all algorithms are very comparable to that of the Monte Carlo search, see Figure 4.27.

Methods based on the CMA-ES perform good overall, especially when the budget is large enough. Especially, BIPOP-CMA-ES, which is the CMA-ES with restarts managing both a small and a large populations, outperforms all algorithms given that the budget is large enough on both noiseless and noisy testbeds.

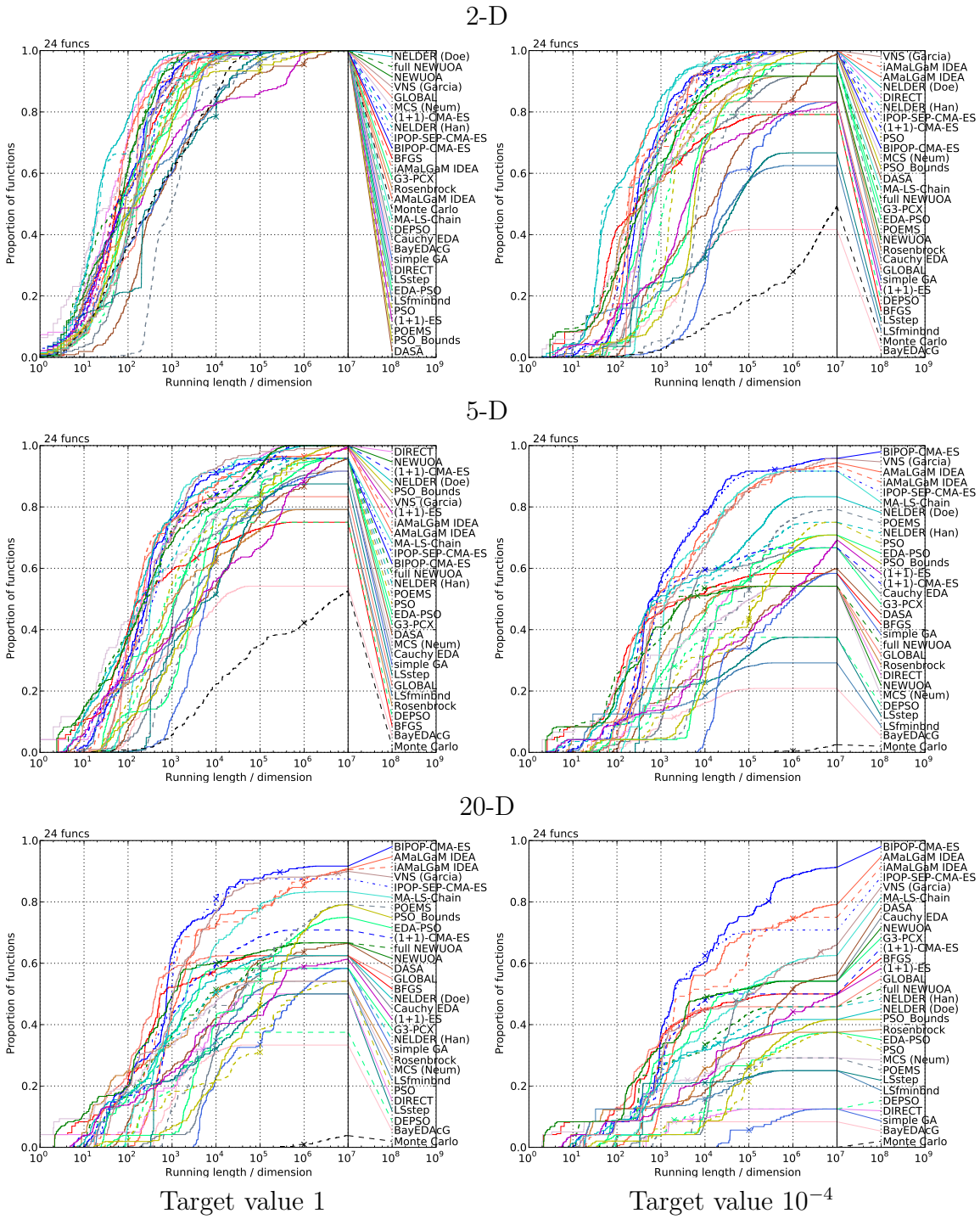


Figure 4.27: Effect of dimensionality (from top to bottom: 2, 5 and 20-D) on the success probability for given running lengths on the noiseless testbed, as shown in empirical cumulative distribution functions of the bootstrap distribution of the running lengths divided by dimension to reach target function values of 1 (left), and 10^{-4} (right column). The median of the number of function evaluations for unsuccessful runs of an algorithm is represented by the single cross on its graph.

Discussion and Perspectives of BBOB 2009 We present here the outcome of the discussions that happened after the BBOB 2009 workshop.

In BBOB 2009, *40-D* was the largest dimension considered which cannot really compare with some real-world BBO problems. Considering problems in larger dimension was suggested as an addition to BBOB 2009.

A process of validation of the results of BBOB 2009 was also requested. The notions of learning, testing and validating constitute the core of experimental machine learning. In supervised machine learning, the goal is to learn a model to accomplish some task which we simplify to labelling samples. An experiment in machine learning can be split in three:

the learning phase consists in giving the samples as well as their labels to the learner to build the model,

in the testing phase, samples are provided for the learner to label, and the number of errors quantify the quality of the learnt model; at this point, one can either go back to the learning phase and try to decrease the number of errors or proceed to the validation phase,

the validation phase consists in giving the learner a number of untouched samples to assess the prediction error.

The validation phase only makes sense if samples of the validation set were never considered in the learning loop. In the case of optimisation, learning and testing is an online process done at each iteration of the optimiser. Validation could be obtained by testing the algorithms on other functions which share the same properties as the ones considered for the learning and testing. In BBOB 2009 this is made possible by considering new instances of the functions in the testbeds for which the different transformations —translations, rotations, ...— are slightly different.

The Crafting Effort CrE was also a point of discussion. The algorithms submitted to BBOB 2009 have a crafting effort equal to zero, except for the GLOBAL algorithm. The crafting effort is provided as a measure of how much an algorithm can be overfitting the functions of the BBOB 2009 testbeds. The definition of CrE cannot account for the case that an algorithm uses *one* set of algorithm parameters specifically designed for a whole BBOB 2009 testbed —which would result in a crafting effort of zero.

The availability of the data of BBOB 2009 was a concern which we are still reflecting on. More specifically, formatting the experimental data in an informative way is an open issue. This brought out the idea to host our experimental framework on a dedicated server but this still is not an immediate concern.

Though some features of the testbeds may have to be further developed, participants of BBOB 2009 agreed that the results of BBOB 2009 could simply be extended by allowing new algorithm entries to be added over time.

4.5 Overall Summary and Discussion

We have presented two successive benchmarking approaches which compare the results of a number of state-of-the-art algorithms from the fields of operational research, global optimisation and evolutionary computation. These algorithms were tested on selected functions that feature some key properties known to make optimisation problems difficult. This allowed to demonstrate some strong and weak points of algorithms. Also the functions studied were scalable, therefore the algorithms were tested in many search space dimensions. The summary of our results on the two benchmarks are presented in Sections 4.3.3 and 4.4.4.

To quantitatively assess a performance in our experimental set-up, it is necessary to have at least a successful event. In this respect, we have seen that *restart strategies* are a great asset. Indeed, the simple strategy that consists in starting the algorithm anew from another random location in the search space improves the success probability of the algorithm given that the number of restarts is large enough. The performance measures that we used accounted for the unsuccessful runs: the success probability is accounted in SP1, whereas the ERT considers the costs of unsuccessful runs.

A particular attention has been brought to *reproducibility* in our experiments and especially in BBOB 2009. Participants were required to provide their experimental results and incited to provide the source code of their experiments, though providing source code may not always be possible and guarantee reproducibility.

Benchmarking is about: 1. defining an experimental set-up that allows comparison and then, 2. presenting comparison results which point out the differences or similarities between the tested algorithms. To extend our study on non-separability,

ill-conditioning and non-convexity we provided more test functions, better selected because these functions display more of the properties that makes optimisation difficult which can be encountered in real-world problems such as noise or multi-modality.

A large part of our work was designing ways to present the results. The figures showing our results were presented to the participants of the BBOB 2009 in addition to the tables and figures of single algorithms included in the workshop papers. In addition to these, comparison tables are being designed and are to be put online. The issue of presenting the multitude of comparison results can be addressed for instance in interactive ways, but this is not in the focus of our study.

Chapter 5

Software: COCO

Contents

The COmparison of Continuous Optimisers (COCO) software[1] is a benchmarking software to render easier experiments in the field of continuous optimisation. A post-processing module generates tables and figures to be included in a research paper template presenting all results.

[1] Available at http://coco.gforge.inria.fr

The GECCO 2009 workshop named Black-Box Optimisation Benchmarking (BBOB 2009)[2] used the COCO software for the generation of the results in all submitted papers, resulting in thirty-eight accepted workshop papers presenting results of state-of-the-art algorithms. Though we will refer to the work provided for the GECCO 2009 workshop, see [Hansen et al., 2009a], Section 4.4, and BBOB 2009 software documentation[3], here we present the COCO software from a more generic perspective.

The COCO software provides:

1. a single generic function interface to the benchmark functions, coded in MATLAB/GNU OCTAVE and C,
2. the experimental framework, centred around the interface function fgeneric,
3. the PYTHON post-processing module bbob_pproc,
4. LaTeX templates to generate papers, and
5. the corresponding documentation.

The practitioner in BBO who wants to benchmark one or many algorithms on the BBOB 2009 testbeds has to download COCO, interface the algorithms to call the test functions in the testbed and use the post-processing tools. The most substantial part is to render the algorithms considered compatible with our software implementation.

Many routines are provided to process the collected data, a command-line interface is provided to generate all the tables and figures presented in the BBOB 2009 workshop papers.

We describe the different steps for obtaining a complete workshop paper for an algorithm, thus allowing us to present the architecture of COCO. We also present additional facilities implemented for the comparison of the results of the many algorithms submitted. Section 5.1 presents the experimental framework software used to generate benchmarking data. Section 5.2 describes the post-processing facilities of COCO, namely the PYTHON package bbob_pproc. Section 5.3 briefly describes the process of compiling a paper regrouping all the post-processed results. Finally, we discuss our implementation of COCO in Section 5.4.

[2] More information at http://coco.gforge.inria.fr/doku.php?id=bbob-2009 and http://www.sigevo.org/gecco-2009/workshops.html#bbob

[3] Available at: http://coco.gforge.inria.fr/doku.php?id=bbob-2009-downloads

Listing 5.1: MY_OPTIMIZER.m: Monte Carlo search in MATLAB. At each iteration, 200 points are sampled and stored in a matrix of size DIM × 200 so as to reduce loops and function calls within MATLAB and therefore improve its efficiency

```
1  function MY_OPTIMIZER(FUN, DIM, ftarget, maxfunevals)
2  % MY_OPTIMIZER(FUN, DIM, ftarget, maxfunevals)
3  % samples new points uniformly randomly in [-5,5]^DIM
4  % and evaluates them on FUN until ftarget of maxfunevals
5  % is reached, or until 1e8 * DIM fevals are conducted.
6  % Relies on FUN to keep track of the best point.
7
8    maxfunevals = min(1e8 * DIM, maxfunevals);
9    popsize = min(maxfunevals, 200);
10   for iter = 1:ceil(maxfunevals/popsize)
11     feval(FUN, 10 * rand(DIM, popsize) - 5);
12     if feval(FUN, 'fbest') < ftarget  % task achieved
13       break;
14     end
15     % if useful, modify more options here for next start
16   end
```

5.1 Experimental Framework Software

The experimental framework software mainly consists in the implementation of the methodology presented in Section 4.4 and [Hansen et al., 2009a]. The software is centred on the interface function, fgeneric. At this date, the interface function fgeneric is implemented in C and MATLAB/GNU OCTAVE. We describe the guidelines of the architecture of fgeneric here.

We also describe the format of the output data files and the content of the files as they are written by fgeneric. These files are to be analysed with the provided post-processing tools that are described further down.

5.1.1 Running Experiments

To display an example of the use of fgeneric, we provide two example scripts. Executing the MATLAB scripts provided in Listings 5.2 and 5.3 results in testing an algorithm —which is MY_OPTIMIZER in the examples, see Listing 5.1— on the noiseless testbed of BBOB 2009 and displaying measures of the time complexity of an algorithm respectively. These scripts are also provided in C. In Listing 5.2, lines 6 to 10 set

Listing 5.2: `exampleexperiment.m`: script for benchmarking MY_OPTIMIZER, see Listing 5.1, for BBOB 2009 on the noiseless function testbed in MATLAB/GNU OCTAVE

```
% runs an entire experiment for benchmarking MY_OPTIMIZER
% on the noise-free testbed. fgeneric.m and benchmarks.m
% must be in the path of Matlab/Octave
% CAPITALIZATION indicates code adaptations to be made

addpath('PUT_PATH_TO_BBOB/matlab');  % should point to fgeneric.m etc.
datapath = 'PUT_MY_BBOB_DATA_PATH';  % different folder for each experiment
opt.algName = 'PUT ALGORITHM NAME';
opt.comments = 'PUT MORE DETAILED INFORMATION, PARAMETER SETTINGS ETC';
maxfunevals = '20 * dim';  % SHORT EXPERIMENT, takes overall three minutes

more off;  % in octave pagination is on by default

t0 = clock;
rand('state', sum(100 * t0)); % initialises the pseudo-random generator
                              % in MY_OPTIMIZER

for dim = [2,3,5,10,20,40]  % small dimensions first, for CPU reasons
  for ifun = benchmarks('FunctionIndices')  % or benchmarksnoisy(...)
    for iinstance = [1:5, 1:5, 1:5]  % first 5 fct instances, three times
      fgeneric('initialize', ifun, iinstance, datapath, opt);

      MY_OPTIMIZER('fgeneric', dim, fgeneric('ftarget'), eval(maxfunevals));

      disp(sprintf(['  f%d in %d-D, instance %d: FEs=%d,' ...
                    ' fbest-ftarget=%.4e, elapsed time [h]: %.2f'], ...
                   ifun, dim, iinstance, ...
                   fgeneric('evaluations'), ...
                   fgeneric('fbest') - fgeneric('ftarget'), ...
                   etime(clock, t0)/60/60));
      fgeneric('finalize');
    end
    disp(['      date and time: ' num2str(clock, ' %.0f')]);
  end
  disp(sprintf('---- dimension %d-D done ----', dim));
end
```

Listing 5.3: `exampletiming.m`: script for measuring the time complexity of MY_OPTIMIZER, see Listing 5.1, for BBOB 2009 in MATLAB/GNU OCTAVE

```
% runs the timing experiment for MY_OPTIMIZER. fgeneric.m
% and benchmarks.m must be in the path of MATLAB/Octave

addpath('PUT_PATH_TO_BBOB/matlab');  % should point to fgeneric.m etc.

more off;  % in octave pagination is on by default

timings = [];
runs = [];
dims = [];
for dim = [2,3,5,10,20,40]
  nbrun = 0;
  ftarget = fgeneric('initialize', 8, 1, 'tmp');
  tic;
  while toc < 30  % at least 30 seconds
    MY_OPTIMIZER(@fgeneric, dim, ftarget, 1e5);  % adjust maxfunevals
    nbrun = nbrun + 1;
  end  % while
  timings(end+1) = toc / fgeneric('evaluations');
  dims(end+1) = dim;    % not really needed
  runs(end+1) = nbrun;  % not really needed
  fgeneric('finalize');
  disp([['Dimensions:' sprintf(' %11d ', dims)]; ...
        ['      runs:' sprintf(' %11d ', runs)]; ...
        [' times [s]:' sprintf(' %11.1e ', timings)]]);
end
```

variables used by fgeneric. The whole set of experiment on the noiseless testbed is done by looping over the lines 18 to 36.

The function fgeneric outputs the results of the experiments, also it provides a single interface to any of the test functions of the BBOB 2009 testbeds. Once fgeneric is loaded into memory, the initialisation process, see line 21 in Listing 5.2, the command fgeneric('initialize', ...) in MATLAB, sets all variables internal to fgeneric: the test function considered, the instance considered, the output directory. After the initialisation, calls to fgeneric evaluate the chosen test function at the point $\boldsymbol{x}$ given as input argument. In the example, calls to fgeneric are done in line 11 of Listing 5.1. The necessary finalisation process, done at the end of a single run for instance in line 31 in Listing 5.2, is described further down in Section 5.1.2.2.

In Listing 5.2, the function f_8 is tested in 2, 3, 5, 10, 20, and 40-D. The `while` loop from line 15 to 18 make the runs last thirty seconds.

5.1.2 Organisation of the Output Data

The output from one *experiment*, consisting of `Ntrials` runs on a given objective function, are contained in a folder whose path is specified by the user. The output consists of an entry in an index file and outputs in two data files, automatically created if necessary, in a fixed folder structure that we describe below. The file extensions are '`*.info`' for the index file and '`*.dat`', '`*.tdat`' for the data files. An example of the folder/file structure is presented in Figure 5.1.

5.1.2.1 Index File

The index file contains meta-information on the optimisation runs and the location of the corresponding data files. The default filename prefix '`bbobexp`' is appended with the function identifier and the extension '`.info`'. An entry in the index file is made of three lines (output format is specified in brackets):

- first line - function identifier (%d), search space dimension (%d), precision to reach (%4.3e) and the identifier of the used algorithm (%s)
- second line - comments of the user (*e.g.* important parameter or used internal methods)

```
↪ container_folder
      ↪ bbobexp_f1.info
      ↪ data_f1
        ↪ bbobexp_f1_DIM5.dat
        ↪ bbobexp_f1_DIM5.tdat
        ↪ bbobexp_f1_DIM10.dat
        ↪ bbobexp_f1_DIM10.tdat
      ↪ bbobexp_f2.info
      ↪ data_f2
        ↪ bbobexp_f2_DIM5.dat
        ↪ bbobexp_f2_DIM5.tdat
↪ container_folder2
↪ ...
```

Figure 5.1: Example data file structures obtained with fgeneric.

```
funcId = 12, DIM = 5, Precision = 1.000e-08, algId = 'ALG-A'
% parameterA = 2, parameterB = 3.34, ...
data_f12\test_f12_DIM5.dat, 1:387|-2.9e-009, 2:450|-2.8e-009, 3:422|-2.1e-009, data_f12\test-01_f12_DIM5.dat, 1:5000000|1.8e-008, ...
funcId = 12, DIM = 10, Precision = 1.000e-08, algId = 'ALG-A'
% parameterA = 2, parameterB = 3.34, ...
data_f12\test1_f12_DIM10.dat, 1:307|-8.6e-008, 2:321|-3.5e-008, ...
...
```

Figure 5.2: Example of an index file

- third line - relative location and name of data file(s) followed by a colon and information on a single run: the instance of the test function, final number of function evaluations, a vertical bar and the final best function value minus target function value.

All entries in the first line and third lines are separated by commas. An example of an index file is given in Figure 5.2. An entry of the index file is written at the start of the first sample run for a given function and dimension.

5.1.2.2 Data Files

A data file contains the numerical output of an optimisation run on a given objective function. The content of the data file is given in the following. Data files are placed in sub-folders at the location of their corresponding index file. At the start of each sample run, a 'header' line with the denomination of each column of the output data is appended to the data file:

- function evaluation
- noise-free fitness - Fopt (and its value)
- best noise-free fitness - Fopt
- measured fitness
- best measured fitness
- x1, x2, ... (one column for each dimension)

These denominations are explained right after. Fopt is the optimum of the test function considered. Lines of output data written below the 'header' line have the following space-separated values:

```
% function evaluation | noise-free fitness - Fopt (6.671000000000e+01) | best noise-free fitness - Fopt | measured fitness | best
measured fitness | x1 | x2 |...
1 +9.324567891e+05 +9.324567891e+05 +1.867342122e+06 +1.867342122e+06 +4.2345e+01 ...
2 +9.636565611e+05 +9.324567891e+05 +8.987623162e+05 +8.987623162e+05 +3.8745e+01 ...
...
31623 9.232667823e+01 9.576575761e+01 -6.624783627e+01 -1.657621581e+02 +5.1234e-02 ...
32478 1.000043784e+02 9.576575761e+01 -4.432869272e+01 -1.657621581e+02 +3.8932e-02 ...
35481 ...
...
```

Figure 5.3: Example of a data file

- recent number of function evaluation (%d),
- recent noise-free function value minus the optimum (%+10.9e),
- best noise-free function value so far minus optimum (%+10.9e),
- recent measured (noisy) function value (%+10.9e),
- best measured (noisy) function value so far (%+10.9e),
- subsequent values are the i-th, $i = 1, 2, \ldots, DIM$, object parameter of the best so far noise-free function value (%+5.4e).

The output data assumes that the optimum value is known. Also the noiseless value of a solution vector is known, even in the case of noisy functions. The fact that the optimum value and the noiseless value of solution vectors are known is crucial to the implementation of bbob_pproc. An example of the content of a data file is given in Figure 5.3.

Each entry in the index files is associated to at least two data files: one for the function value-aligned data and another for the number of function evaluations-aligned data, respectively the vertical and horizontal views, see Appendix B. The data file names are identical except for the file extension being '`*.dat`' and '`*.tdat`' respectively.

The writing to the function value-aligned data file happens only each time the noise-free function value minus the optimum function value is less than $10^{i/5}$, for all integer i, for the first time —this difference is not provided by fgeneric to the tested algorithm. For instance, if the sequence of the best noise-free function values obtained minus the optimum is {**7.72**, 20.8, 12.7, 9.98, 13.6, 13.8, 17.1, 9.86, **3.42**, 2.97, **0.47**, 0.76, 0.40, **0.04**, ...}, the written sequence of function values would be: **7.72**; **3.42** which is the first value smaller than $10^{4/5} \approx 6.31$, the closest from below to 7.72 in

the series of the $10^{i/5}$; **0.47** which is the first value smaller than $10^{2/5} \approx 2.51$, the closest from below to 3.42 in the series of the $10^{i/5}$; **0.04**; ...

The writing to the number of function evaluations-aligned data file happens:

- each time the function evaluation number is equal to $\lfloor 10^{i/20} \rfloor$ for at least one $i = 1, 2, \ldots$ This means, that writing happens after about 12.2% additional function evaluations have been conducted. In particular the first 8 evaluations are written and also evaluations ..., 89, 100, 112, 125, 141,..., 707, 794, 891, 1000, 1122,...
- when any termination criterion is fulfilled (writing the recent evaluation and the current best so far values)

In MATLAB, the function evaluations-aligned data file is modified by the finalisation process of fgeneric: before closing the file the finalisation process inserts the data of the best-ever fitness value and appends that of the final function evaluation. The function evaluation for which the best-ever fitness value is inserted in the function evaluations-aligned data file if it was not already written before. The insertion of the data of the best-ever fitness value is not done in C because of technical reasons.

The default prefix for the data file names is also '`bbobexp`'. The function identifier and the dimension of the object parameters are appended to this prefix. All data files are saved in sub-folders `data_fX`, where `X` is the function identifier, located at the same location as their index file.

5.1.3 Resuming Experiments

In the case of an interrupted experiment, no automated process is provided to resume the run aborted when the interruption occurred or resume the whole experiment. Though running the post-processing on this interrupted set of data would not raise any error, one might want to remove the data corresponding to the interrupted run instead of redoing the whole experiment. We propose a manual process to remove the data corresponding to the interrupted run before redoing it.

1. Find the last modified `.info` file. The function identifier and dimension where the experiment was aborted, are given in the third to last line of the file. For instance:

```
funcId = 13, DIM = 40, Precision = 1.000e-08, algId = 'my optimizer'
% all default parameters
data_f13/bbobexp_f13_DIM40.dat, 1:5387|-4.4e-09, 2:5147|-3.9e-09, 3
```

The last line indicates the name of one of the corresponding data file. The last number in the last line is the function instance number of the unfinished run.

2. Remove the last characters, in the above example, ", 3" from the last line of the index file. If it was the first entry that was interrupted remove the last three lines of the index file.

3. Remove the respective data of the unfinished last trial which starts with the "header" line described above. This has to be done for *both* data files with the extensions `.dat` and `.tdat`, in our example `data_f13/bbobexp_f13_DIM40.dat` and `data_f13/bbobexp_f13_DIM40.tdat`.

4. Modify the experiment script to restart the experiment from this very function instance.

5.2 Post-Processing the Experimental Data

The PYTHON post-processing tool, called bbob_pproc in BBOB 2009 generates image files and LaTeX tables from the raw experimental data obtained as described previously in Section 5.1.1.

The entire post-processing tool requires that PYTHON is installed on your machine. The minimal software requirements for using the post-processing tool are Python (2.5.2), Matplotlib (0.91.2) and Numpy (1.0.4). The installation of the software is described in Appendix C.

5.2.1 Overview of the bbob_pproc Package

We present here the content of the latest version of the bbob_pproc package (3.6beta, revision 1585 of the COCO repository).

`run.py` is the main interface of the package that calls the different routines listed below,

`pproc.py` defines the classes DataSetList and DataSet which are the main data structures that we use to gather the experimental raw data,

`dataoutput.py` contain routine to output instances of DataSet,

`ppfigdim.py`, `pptex.py`, `pprldistr.py` are used to produce figures and tables that we describe further down,

`readalign.py`, `bootstrap.py` contain routines for the post-processing of the raw experimental data.

The class DataSetList inherits from list and is a collection of instances of the class DataSet. An instance of the class DataSet represents the unit component of experimental data: it is associated to the results of a single algorithm using a given set of parameters on a single problem. For BBOB 2009, a problem corresponds to an objective function in a given dimension. Thus, an instance of DataSet aggregates the information of `Ntrials` runs which are sorted in the order of the problem instances designated by their instance number, stored in the attribute itrials of DataSet.

The data are stored in the form of numpy.ndarray, which is a data structure representing an array. To explain the representation of the data, we will refer to horizontal and vertical cuts in the Figure 5.4 which correspond to a fixed-target or a fixed-budget scenario, see Appendix B. From this point on, we will refer to the function values offset by the optimal function value f_{opt} of the problem instance considered instead of the absolute function values. In the case of noisy function, we will refer to the noise-free function value. Please note that the optimum of the offset functions values is zero. To represent all the raw data corresponding to an instance of DataSet, mainly four arrays are used:

funvals, each line of the array represents a vertical cut in the Figure 5.4, the zero-th column giving the number of function evaluations (x-position) of the cut; the subsequent columns, sorted in the order of the problem instances processed, correspond to the best function values obtained after this number of function evaluations,

finalfunvals is a single-line array containing the final function values obtained,

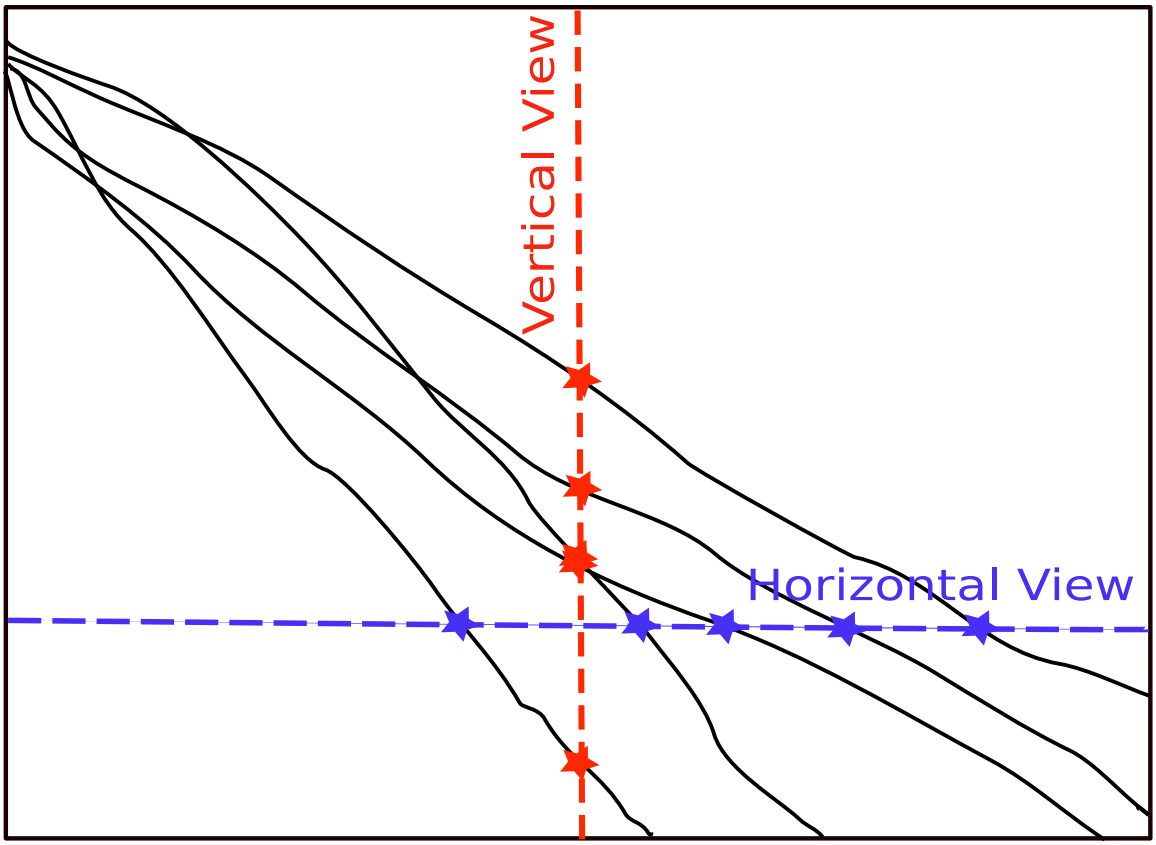


Figure 5.4: Horizontal view or fixed-target scenario and vertical view or fixed-budget scenario for convergence graphs, the axes represent function values versus time [Hansen et al., 2009a]

evals is the counterpart of funvals, but with each line representing a horizontal cut, the zero-th column gives the function value (y-position) of the cut, the subsequent columns numbers of function evaluations to surpass the aforementioned function value,

maxevals is a single-line array of the maximum number of function evaluations.

Routines from `readalign.py` are used to generate the previously mentioned arrays. One of these routines is used when for a vertical cut, meaning that we consider a number of function evaluations, a function value is not available. This might occur in the generation of funvals[4]. If a function value is not available, the last function value obtained for this particular instance is repeated.

The generation of evals differs at least by two things. First, the function values giving us the horizontal cuts used are given by the sequence of $10^{i/5}$, with $i = 1, 2, \ldots$, and cannot be obtained from the raw data because of numerical round-off errors. Second, if we consider a function value, in the case that it is never surpassed before the maximum number of function evaluations for a particular instance, numpy.nan is inserted instead.

We list here the parts of bbob_pproc that are specific to BBOB 2009

- a problem is represented by the many different instances of one function in one dimension,
- an important feature used by the post-processing and more particularly the output part is that the optimum function value f_{opt} is available,
- the output modules which we describe below.

5.2.2 Performance Assessment of Algorithms

We recall that we use the expected running time ERT for the algorithm to surpass a target function value, see Section 4.4.2.2. A first part of the post-processing consists in computing the ERT for each line of evals.

We describe the output of the different modules of bbob_pproc that were already mentioned by providing example output. These modules all feature a main method

[4]The writing only occur if the number of function evaluations is equal to $\lfloor 10^{i/20} \rfloor$, with $i = 1, 2, \ldots$

Table 5.1: Example table obtained with `pptex.py`. Shown are, for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\text{opt}} + \Delta f$ (ERT); the **10%**-tile and **90%**-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value (RT_{succ}). If $f_{\text{opt}} + \Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial.

	f_{19} in 5-D, N=15, mFE=37734					**f_{19} in 20-D**, N=15, mFE=255570				
Δf	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.7e3	1.2e3	2.2e3	1.7e3	3	1.2e6	7.0e5	3.5e6	2.5e5
1	12	2.2e4	1.7e4	2.8e4	1.7e4	0	*12e+0*	*71e−1*	*15e+0*	1.6e5
1e − 1	1	4.3e5	2.1e5	> 4e5	2.8e4	.	.	.	.	.
1e − 3	0	*62e−2*	*22e−2*	*12e−1*	1.8e4	.	.	.	.	.
1e − 5	.	.	.	.	.	.	.	.	.	.
1e − 8	.	.	.	.	.	.	.	.	.	.

that take as input an instance of DataSetList that lists the instances for which we want the output.

`ppfigdim.py` is used to generate figures of the expected run times, ERT, versus the dimension, see for instance Figure 5.5,

`pptex.py` is used to generate LaTeX tables, see for instance Table 5.1,

`pprldistr.py` is used to generate the figures of empirical cumulative distribution functions of the run lengths and of the final function values, see for instance Figure 5.6.

Details are provided in the caption of Figures 5.5, 5.6 and Table 5.1.

5.2.3 Comparison of Algorithms

Additional features of bbob_pproc that are still in development were used to present the comparison results of Section 4.4.3. These features are gathered in the modules `determineFtarget.py`, `ppperfprof.py` and `pptables.py`.

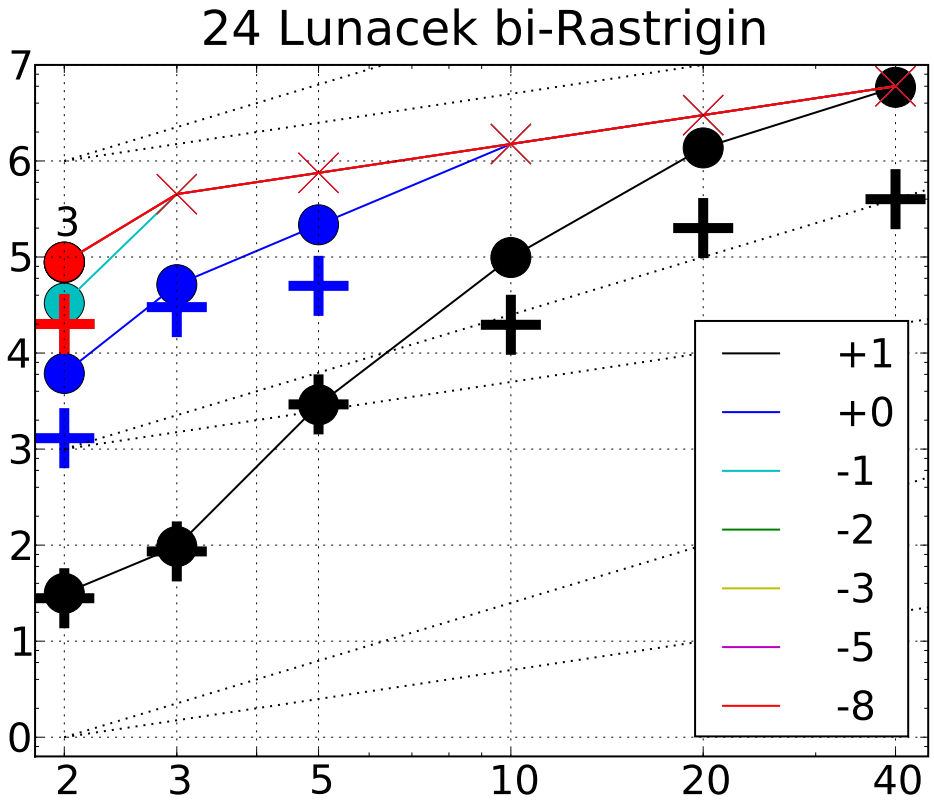


Figure 5.5: Example figure output by `ppfigdim.py`. Expected Running Time (ERT, ●) to reach $f_{\text{opt}} + \Delta f$ and median number of function evaluations of successful trials (+), shown for $\Delta f = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ (the exponent is given in the legend) versus dimension. f_{opt} denotes the optimal function value. Crosses (×) indicate the total number of function evaluations. Numbers above ERT-symbols indicate the number of successful trials. Annotated numbers on the ordinate are decimal logarithms. Additional grid lines show linear and quadratic scaling

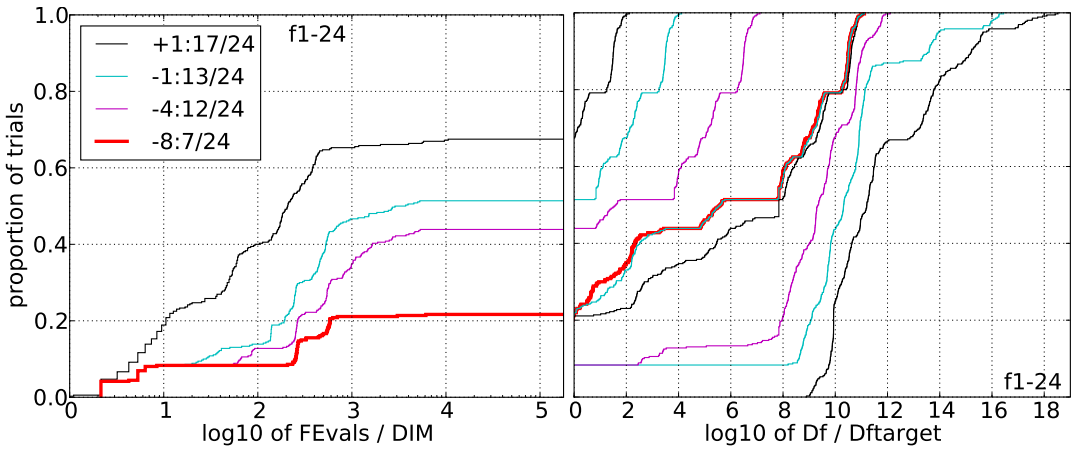


Figure 5.6: Example figure output by `pprldistr.py`. Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplot) or versus Δf (right subplot). The thick red line represents the best achieved results. Left subplot: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^k (upper left lines in continuation of the left subplot), and best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D \ldots$ function evaluations (from right to left cycling black-cyan-magenta). The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, D and DIM denote search space dimension, and Δf and Df denote the difference to the optimal function value.

5.2.4 Using the bbob_pproc Package

To perform the post-processing on the experimental data obtained as described previously, the bbob_pproc package need to be downloaded[5] and un-archived. Then, to post-process the data, the data folder `DATAPATH` containing all data generated by the experiments needs to be in the current working directory before executing the following command:

```
python path_to_postproc_code/bbob_pproc/run.py DATAPATH
```

from a shell[6], the folder `path_to_postproc_code` is the one where the provided post-processing software was un-archived.

The above command create the folder with the default name `ppdata` in the current working directory, which contain the post-processed data in the form of figures and LaTeXfiles for the tables. This process might take a few minutes.

To run the post-processing directly from a Python shell, the following commands need to be executed:

```
>>> import bbob_pproc
>>> bbob_pproc.main('DATAPATH')
```

This first command loads bbob_pproc into memory and requires that the path to the package is in the Python search path.

The resulting `ppdata` folder now contains a number of TeX, eps, png files.

Additional help for the bbob_pproc package can be obtained by executing the following command in a shell:

```
python path_to_postproc_code/bbob_pproc/run.py -h
```

In particular, this command describes the additional options for the execution of the post-processing. The code documentation can be found in the folder `path_to_postproc_code/pyc` within the provided software package.

[5] The package can be obtained from http://coco.gforge.inria.fr/doku.php?id=bbob-2009.

[6] Note that in Windows the path separator '\' must be used instead of '/'

5.3 Generating a Paper

`templateBBOBarticle.tex` and `templateBBOBnoisyarticle.tex` are the template LaTeX files that include all the figures and tables presenting the result of an algorithm on the noiseless and noisy testbeds of BBOB 2009. If compiled correctly using LaTeX, it generates documents collecting and organising the output from bbob_pproc. Each of the templates has a given page organisation optimised for the presentation of the results on each testbed.

To compile a document, one needs:

1. to have a working LaTeX distribution[7],
2. to be in the correct working directory (containing the folder `ppdata` that includes all the output from the bbob_pproc),
3. that `templateBBOBarticle.tex`[8], `bbob.bib` and `sig-alternate.cls` are in the working directory (all files are provided with the software),

Then the following commands needs to be executed in a shell:

```
latex templateBBOBarticle
bibtex templateBBOBarticle
latex templateBBOBarticle
latex templateBBOBarticle
```

The document `templateBBOBarticle.dvi` is then generated in the format required for a GECCO workshop paper. An example of the resulting template document obtained by compiling the LaTeX template paper is provided here[9].

[7] http://www.latex-project.org/

[8] or `templateBBOBnoisyarticle.tex` for the noisy testbed of BBOB 2009.

[9] The figures and tables show the data of the Monte Carlo search on the noiseless testbed of BBOB 2009 [Auger and Ros, 2009a].

Black-Box Optimization Benchmarking Template for Noiseless Function Testbed

Draft version [*]

Forename Name

ABSTRACT

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*global optimization, unconstrained optimization*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Benchmarking, Black-box optimization, Evolutionary computation

1. RESULTS

Results from experiments according to [2] on the benchmark functions given in [1, 3] are presented in Figures 1 and 2 and in Table 1.

2. REFERENCES

[1] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, 2009.

[2] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, 2009.

[3] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009.

[*]Camera-ready paper due April 17th.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

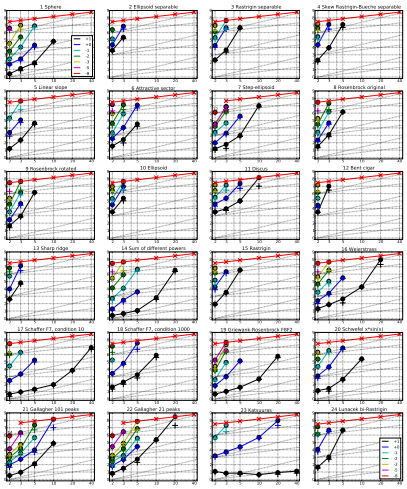


Figure 1: Expected Running Time (ERT, ●) to reach $f_{\mathrm{opt}} + \Delta f$ and median number of function evaluations of successful trials (+), shown for $\Delta f = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ (the exponent is given in the legend of f_1 and f_{24}) versus dimension in log-log presentation. The $\mathrm{ERT}(\Delta f)$ equals to $\#\mathrm{FEs}(\Delta f)$ divided by the number of successful trials, where a trial is successful if $f_{\mathrm{opt}} + \Delta f$ was surpassed during the trial. The $\#\mathrm{FEs}(\Delta f)$ are the total number of function evaluations while $f_{\mathrm{opt}} + \Delta f$ was not surpassed during the trial from all respective trials (successful and unsuccessful), and f_{opt} denotes the optimal function value. Crosses (×) indicate the total number of function evaluations $\#\mathrm{FEs}(-\infty)$. Numbers above ERT-symbols indicate the number of successful trials. Annotated numbers on the ordinate are decimal logarithms. Additional grid lines show linear and quadratic scaling.

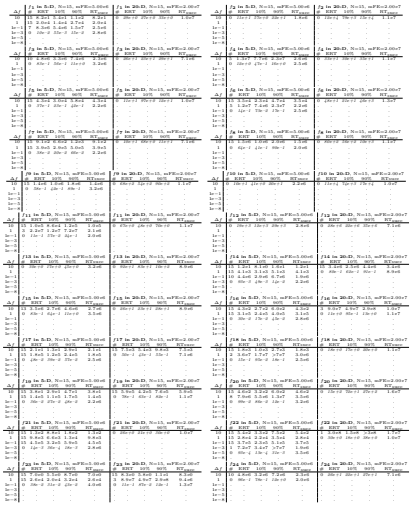


Table 1: Shown are, for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\mathrm{opt}} + \Delta f$ (ERT, see Figure 1); the 10%-tile and 90%-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value ($\mathrm{RT}_{\mathrm{succ}}$). If $f_{\mathrm{opt}} + \Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial. See Figure 1 for the names of functions.

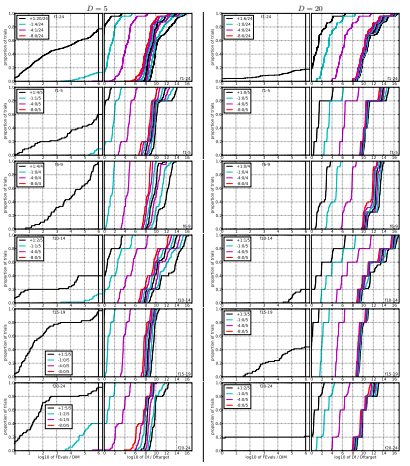


Figure 2: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplots) or versus Δf (right subplots). The thick red line represents the best achieved results. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^k (upper left lines in continuation of the left subplot), and best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D \ldots$ function evaluations (from right to left cycling black-cyan-magenta). Top row: all functions; second row: separable functions; third row: misc. moderate functions; fourth row: ill-conditioned functions; fifth row: multi-modal functions with adequate structure; last row: multi-modal functions with weak structure. The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, D and DIM denote search space dimension, and Δf and Df denote the difference to the optimal function value.

The participants of BBOB 2009 were expected to fill in the template with all of their information, the description of their algorithm and their parameter settings [Hansen et al., 2009a], their source code or a reference to it, their results on the timing experiment. The BibTeX file `bbob.bib` includes the references to the BBOB 2009 experimental set-up and documentation.

5.4 Discussion of our Implementation

We have presented the COCO software which consists of two different pieces of software which are both inscribed in the experimental methodology of BBOB 2009.

A first part is fgeneric which provides a unified experimental framework as well as a single interface function for whole experiments. The use of the single function fgeneric for doing experiments, initialising, and finalising altogether is quite specific to Matlab/Gnu Octave. As is done in the C-code of fgeneric, these processes can be split into different functions. Among the specifications for the development of such experimental framework, we included a way of distributing whole experiments over time and space through the use of the index files and data files: merging together results obtained from multiple work session over time or over multiple computer nodes is possible as long as a single computational process at a time is writing into a given index file. A feature that is lacking for the moment in fgeneric due to the management of all internal variables is the possibility of distributing a *single run* over multiple computer nodes.

A second part of COCO is the Python package bbob_pproc which provide facilities for the post-processing of experimental data. The representation of a unit component of experimental data by instances of DataSet makes sense in our implementation which focuses on aggregating the information from a number of such unit components.

Our goal of using COCO was attained through the results of BBOB 2009. The efforts we have put into our implementation will be prolonged by porting COCO into other programming language, extending the use of COCO to other algorithms, other benchmarks.

Chapter 6

Summary and Perspectives

This manuscript presents our contributions in the context of Black-Box Optimisation (BBO) and evolutionary computation. The key question addressed is that of the comparison of algorithms in order to choose an appropriate algorithm in the face of a BBO problem. To evaluate the performances of algorithms, we consider the notion of *search costs* which we define as the quantity of computation that is required for an algorithm to reach a solution.

This manuscript proposed and put into practice an experimental methodology to compare algorithms on artificial test problems. These comparisons brought to light many useful insights on the behaviour of algorithms with respect to difficulties of optimisation.

6.1 Algorithms for High Dimensional Optimisation Problems

In the context of optimisation, the 'curse of dimensionality' is a difficulty that relates to the exponential increase of the search space as its dimension increases. The curse of dimensionality is a concern when dependencies between the parameters are relevant to solving the problem considered. The Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) successfully addresses the issue of learning dependencies in real-parameter search spaces, which make the CMA-ES a good candidate algorithm in

BBO. The CMA-ES learns all pair-wise dependencies between parameters by updating a covariance matrix for the sample distribution. In learning all pair-wise dependencies, the internal space and time complexity of CMA is at least $\mathcal{O}(n^2)$, where n is the dimension of the search space, which is a limitation on high dimensional problems. We propose to change the update rule of the covariance matrix by putting constraints on the degrees of freedom of the matrix. The sep-CMA-ES algorithm updates only the diagonal of the covariance matrix, which results in a $\mathcal{O}(n)$ time and space complexity. The block-CMA-ES updates only a block-diagonal covariance matrix, the number of blocks and their size provide with a parametrisation of the complexity of the algorithm.

As a consequence to the constraints on the degrees of freedom, adjustments are made to the learning rate parameter used in the update rule of the covariance matrix. The sep-CMA-ES and block-CMA-ES variants have been compared to the CMA-ES on test problems. Also we compared sep-CMA-ES to other variants of CMA-ES from the literature addressing the issue of the time and space complexity. Expectedly on separable functions, we show that the search costs of sep-CMA-ES scale linearly with the dimension of the search space and that block-CMA-ES displays performances which covers the range in-between the performances of CMA-ES and those of sep-CMA-ES.

The sep-CMA-ES and block-CMA-ES are proofs of concept that updating part of the covariance matrix is a possibility in the case of large scale problems. The sep-CMA-ES is also a baseline comparison for algorithms that exploits separability. A surprising result is that even on non-separable functions sep-CMA-ES can perform better than the CMA-ES given that the dimension of the problem is large enough.

On the base of our results we propose a new policy in the face of BBO problems consisting in using sep-CMA-ES for the first $100n/\sqrt{\lambda}$ iterations, where n is the dimension of the search space and λ is the population size, before switching to the standard CMA-ES. We advise to use the separable strategy for what we estimate to be the number of iterations that the learning phase requires so as to benefit from the separable strategy for at least a fraction of the learning time. This strategy ensures good performances on separable functions in exchange for losses that we assume to be minimal on non-separable functions.

Extensions of this work could consists in the study of other CMA-ES variants with

constraints on the covariance matrix. Also we consider testing the sep-CMA-ES and block-CMA-ES on real-world functions.

6.2 Benchmarking

Our study of benchmarking was done in two successive steps, both of them on two different scales. First, we studied a number of stochastic methods from the field of evolutionary computation such as Differential Evolution (DE), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), Particle Swarm Optimisation (PSO), a local search deterministic algorithm BFGS and a novel trust region method NEWUOA. These algorithms were studied with respect to dimensionality, non-separability, ill-conditioning, and non-convexity on a restrained set of test functions. We showed the rotation of the search space could very well affect the performances of algorithms such as PSO, NEWUOA and BFGS. We also showed that methods such as NEWUOA and BFGS are affected by the non-convexity of the fitness function.

The second step of our study was made on a larger scale. We presented the results of the BBO Benchmarking (BBOB) 2009 workshop which featured the results of twenty-nine algorithms on a testbed of twenty-four noiseless functions and thirty noisy functions. We showed some methods such as the Nelder-Mead simplex perform best in 2-D and 3-D, whereas their performances do not scale well in higher dimensions. Some methods perform well for smaller budgets whereas the methods based on the adaptation of the covariance matrix are very well adapted when the budget is larger. Also the BBOB 2009 allowed to display the features of the COCO software.

Many extensions to BBOB 2009 can be considered. Some extensions are obvious like adding functions to our test suites or by adding the performances of other algorithms. Considering problems in even higher dimensions or some well-chosen real-world problems would be conceptually more substantial additions since new issues would arise: would the study of the scalability of algorithms as the dimension increases still be possible with real-world problems? Are fifteen repetitions computationally feasible in higher dimensions?

Conceptually sound, our implementation of COCO can be improved in many ways especially with respect to the objective of providing more researchers with a benchmarking software.

Appendix A

Parameter Identification of DE

Differential Evolution (DE) is an evolutionary algorithm that makes use of a differential mutation which basically adds the weighted difference between two population vectors to a third vector, see Section 2.1.2.8. Many variants of the differential mutation procedure exists. Choosing between these variants and setting the parameter F and CR requires preliminary testing as [Storn and Price, 1997] admits that the results of the algorithm are dependent on the chosen strategy and the choice of parameter.

We test the original code provided by Storn[1] which proposes six strategies [Price et al., 2005]. The DE/x/y notation specifies for x the vector to be mutated, y the number of difference vectors used. The way the strategies are numbered is relevant since it corresponds to the numeration in the code.

1. DE/rand/1 is the original DE as presented in Section 2.1.2.8,
2. DE/local-to-best/1 is a variant where instead of the base vector $\boldsymbol{x}_{i_1}$ being chosen in the population vector, it is chosen to lie between the vector considered and the best vector so far, thus the update of the velocity is written as follows: $\boldsymbol{v}_i = \boldsymbol{x}_i + F(\boldsymbol{x}_{\text{best}} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{i_2} - \boldsymbol{x}_{i_3})$,
3. DE/best/1 with jitter, in this case the base vector is the best vector so far, a quantity uniformly sampled between zero and 10^{-4} is added to the parameter F for the computation of each component of $\boldsymbol{v}_i$, different for each vector $\boldsymbol{v}_i$,

[1] MATLAB code available here: http://www.icsi.berkeley.edu/~storn/code.html

4. DE/rand/1 with per-vector-dither is the original DE with a quantity uniformly sampled between zero and $1 - F$ added to F, different for each vector $\boldsymbol{v}_i$,

5. DE/rand/1 with per-generation-dither is the original DE with a quantity uniformly sampled between zero and $1-F$ added to F, different at each generation,

6. DE/rand/1 either-or algorithm is the original DE but using randomly either the classical differential mutation or a three-point-recombination.

A.1 Experimental Set-up for the Identification of the Parameters of DE

All six strategies are tested on the rotated ellipsoid function with a condition number of a hundred $f_{\text{elli}}(\boldsymbol{x}) = \sum_{i=1}^{n} 100^{\frac{i-1}{n-1}} y_i^2$ in 5, 10 and 20-D. The starting point of the algorithm is uniformly sampled in the range $[-3, 10]^n$. The population sizes considered are 1, 3, 5, 10 and 30 times the dimension of the search space. The maximum number of function evaluations is set to $1000n$ times the population size. The parameter CR is chosen in the range $[0, 1]$, F in $[0.3, 0.9]$. Each experiment is repeated three times.

The performance measure is the average number of function evaluations to reach the target function value 10^{-7}.

A.2 Results and Discussion

Figures A.1 to A.5 present the results of the identification of the parameters of DE on the rotated ellipsoid function in 5-D. Figures A.6 to A.10 correspond to the rotated ellipsoid function in 10-D and Figures A.11 to A.15 in 20-D.

As stated in Section 4.2.4, the ratio between the best and worst performances can be as large as a thousand. Also, the best performances, meaning the smaller number of function evaluations to reach the target function value of 10^{-7}, overall are obtained with CR and F both close to one, though it is worth noting that the number of function evaluations does not behave monotonically as for a given CR when F increases and for a given F when CR increases.

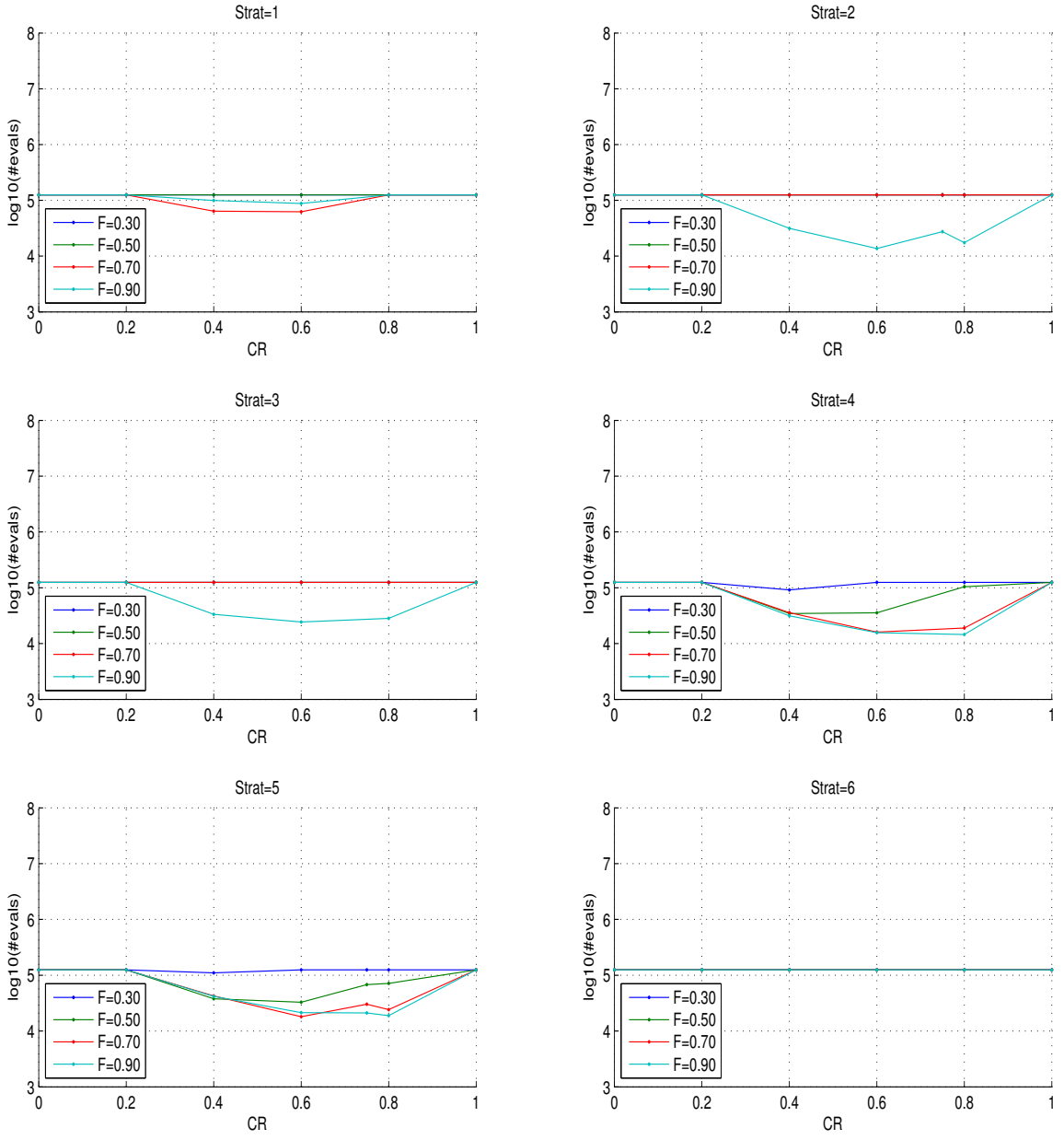


Figure A.1: Parameter identification of DE on the rotated ellipsoid function in 5-D with a population size of one times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

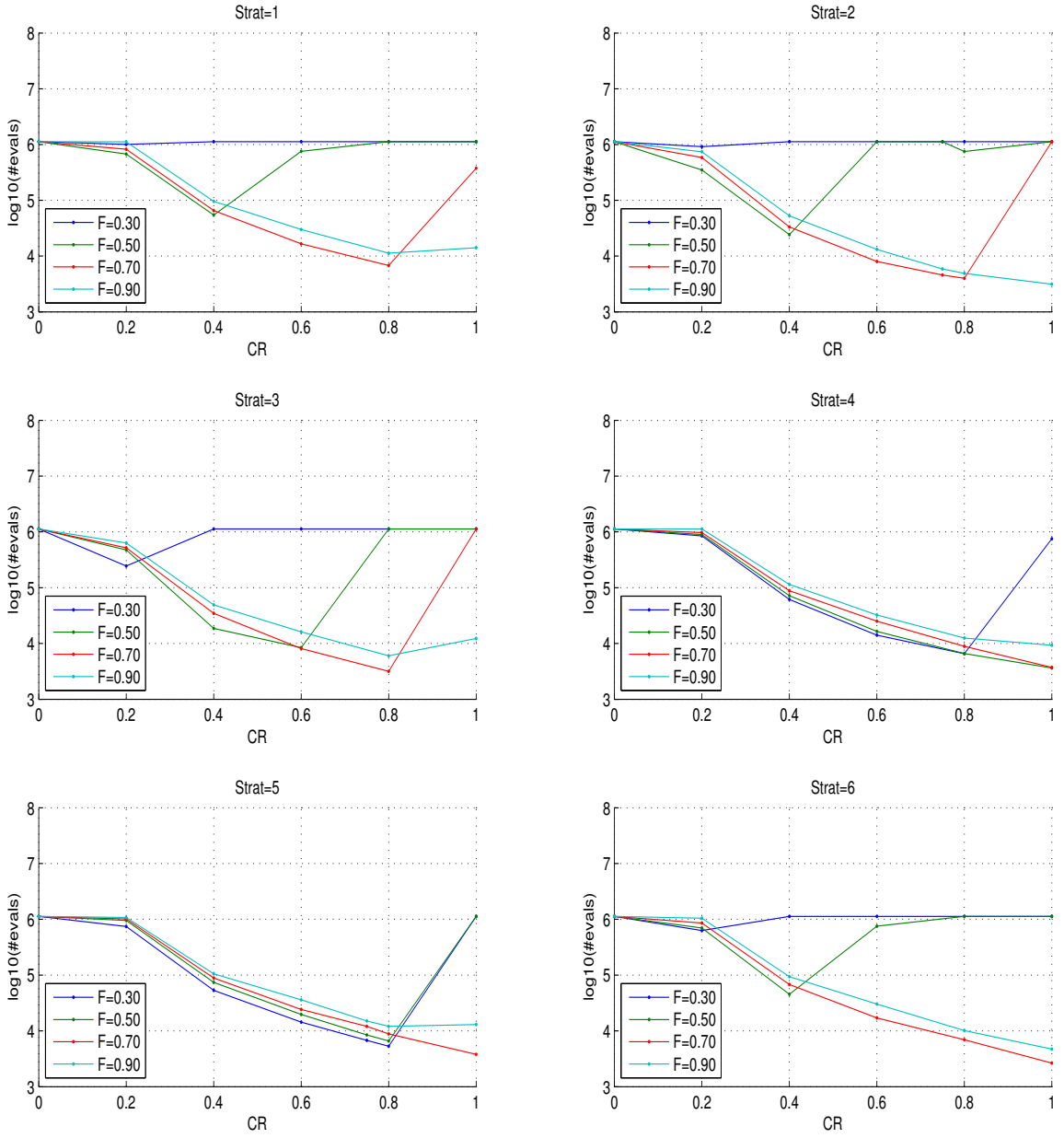


Figure A.2: Parameter identification of DE on the rotated ellipsoid function in 5-D with a population size of three times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

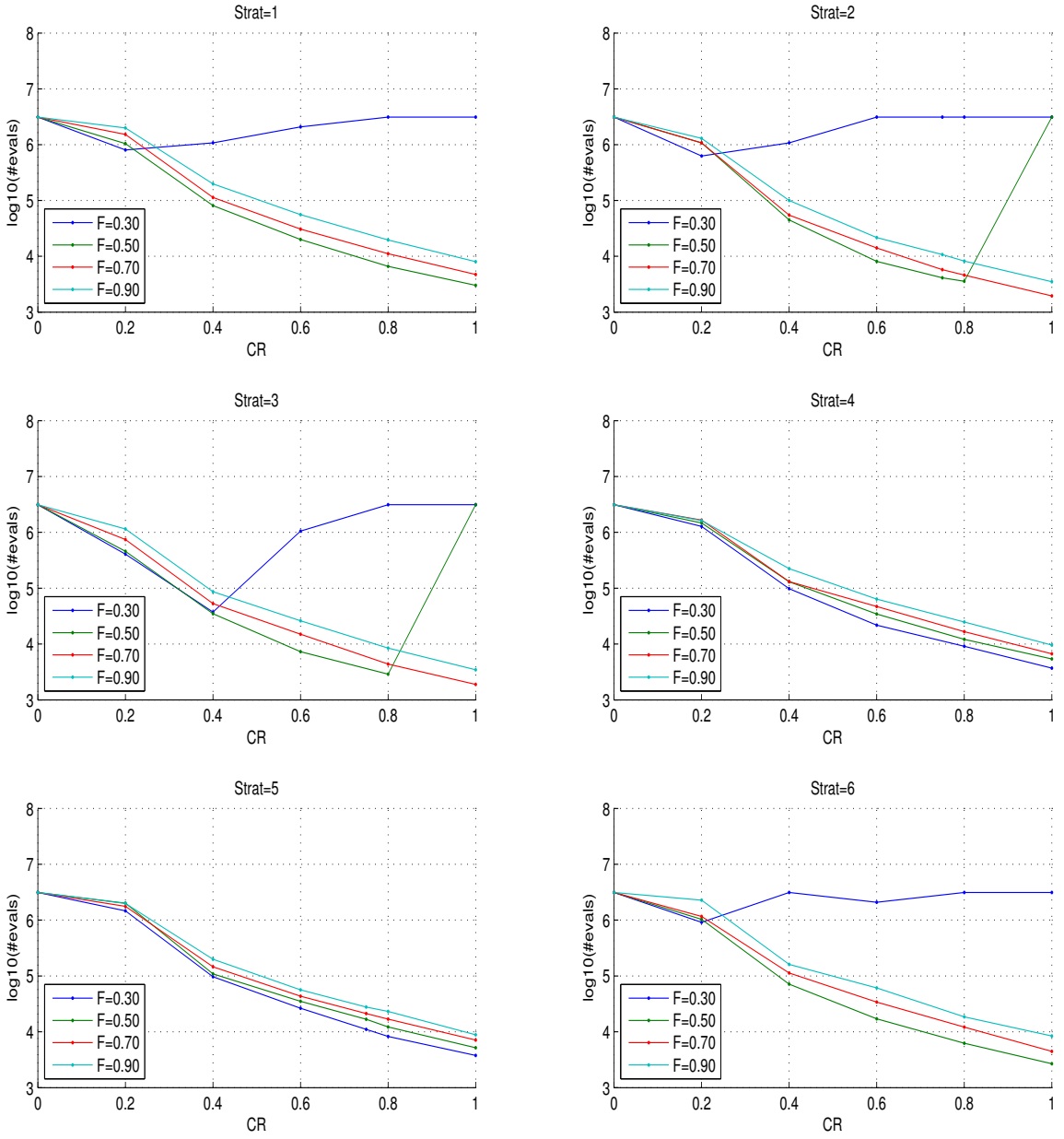


Figure A.3: Parameter identification of DE on the rotated ellipsoid function in 5-D with a population size of five times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

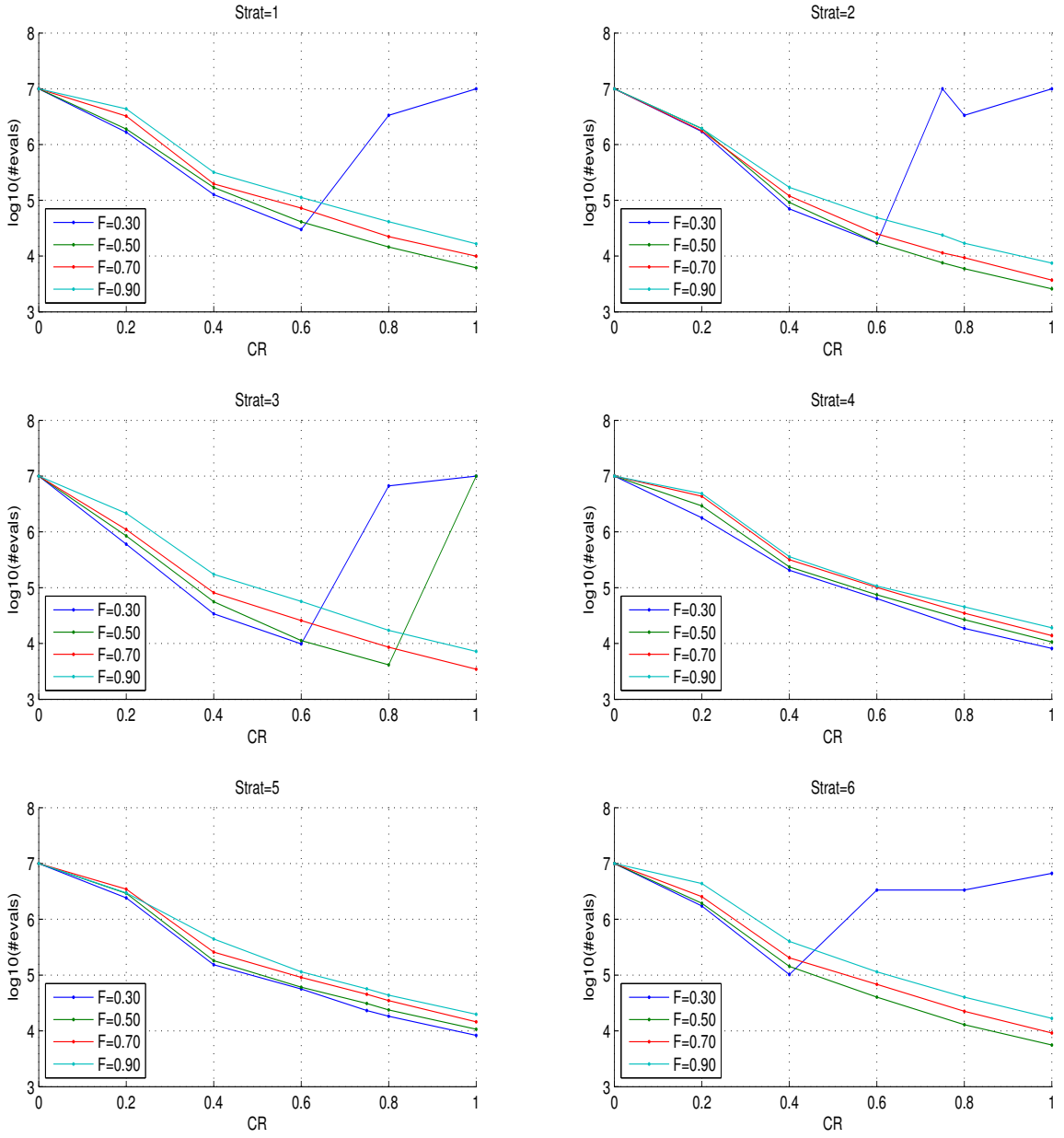


Figure A.4: Parameter identification of DE on the rotated ellipsoid function in 5-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

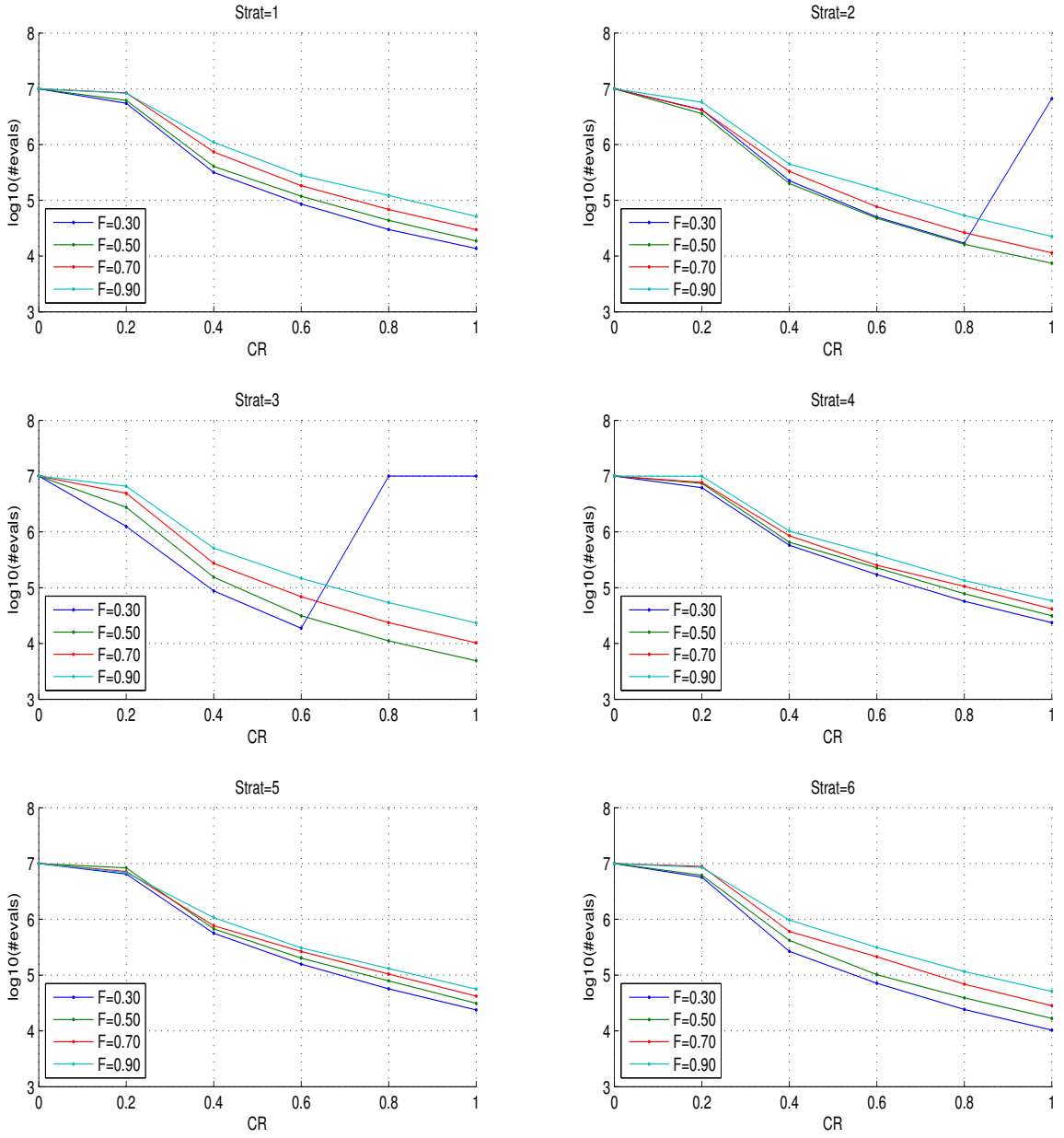


Figure A.5: Parameter identification of DE on the rotated ellipsoid function in 5-D with a population size of thirty times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

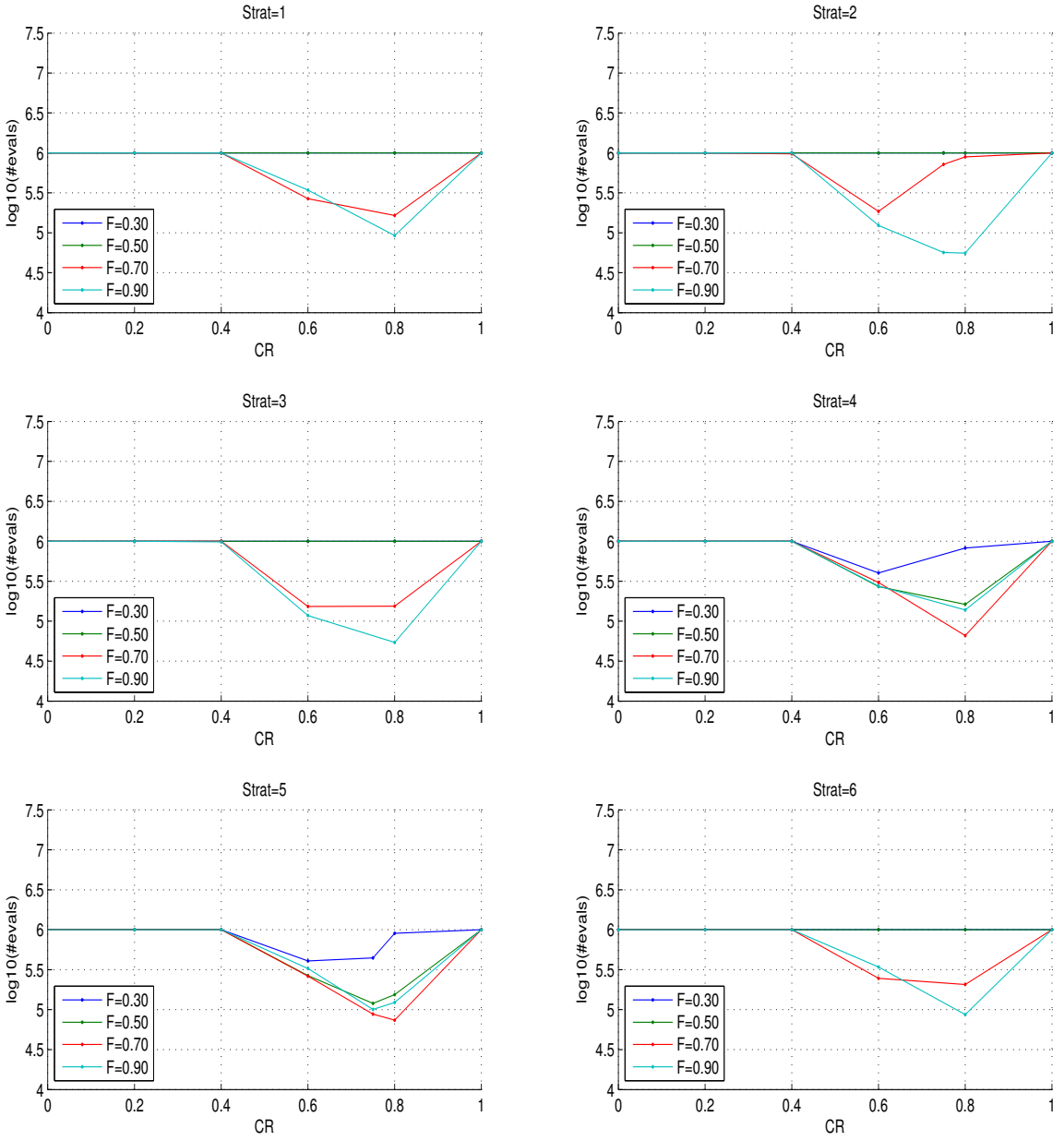


Figure A.6: Parameter identification of DE on the rotated ellipsoid function in 10-D with a population size of one times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

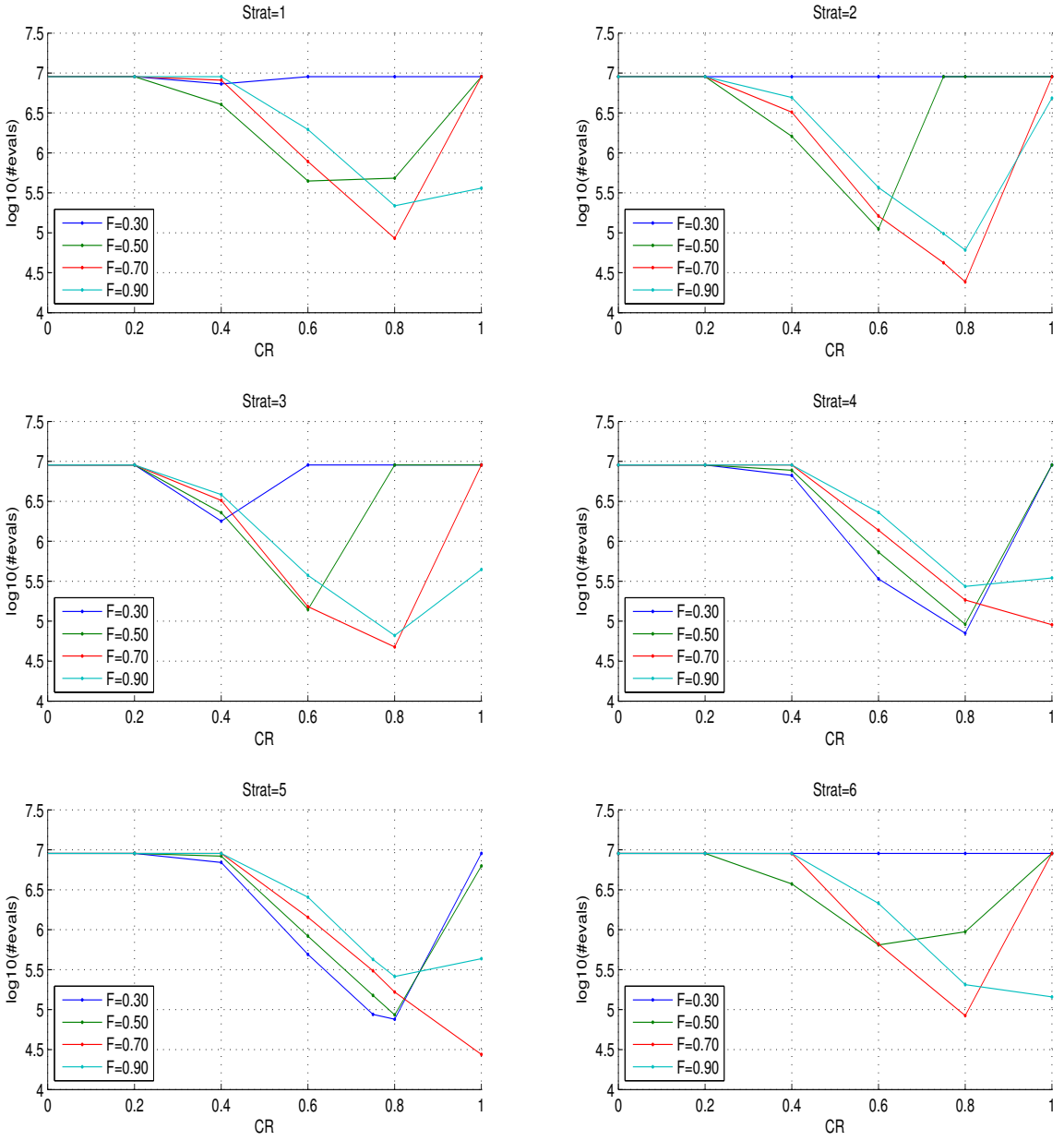


Figure A.7: Parameter identification of DE on the rotated ellipsoid function in 10-D with a population size of three times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

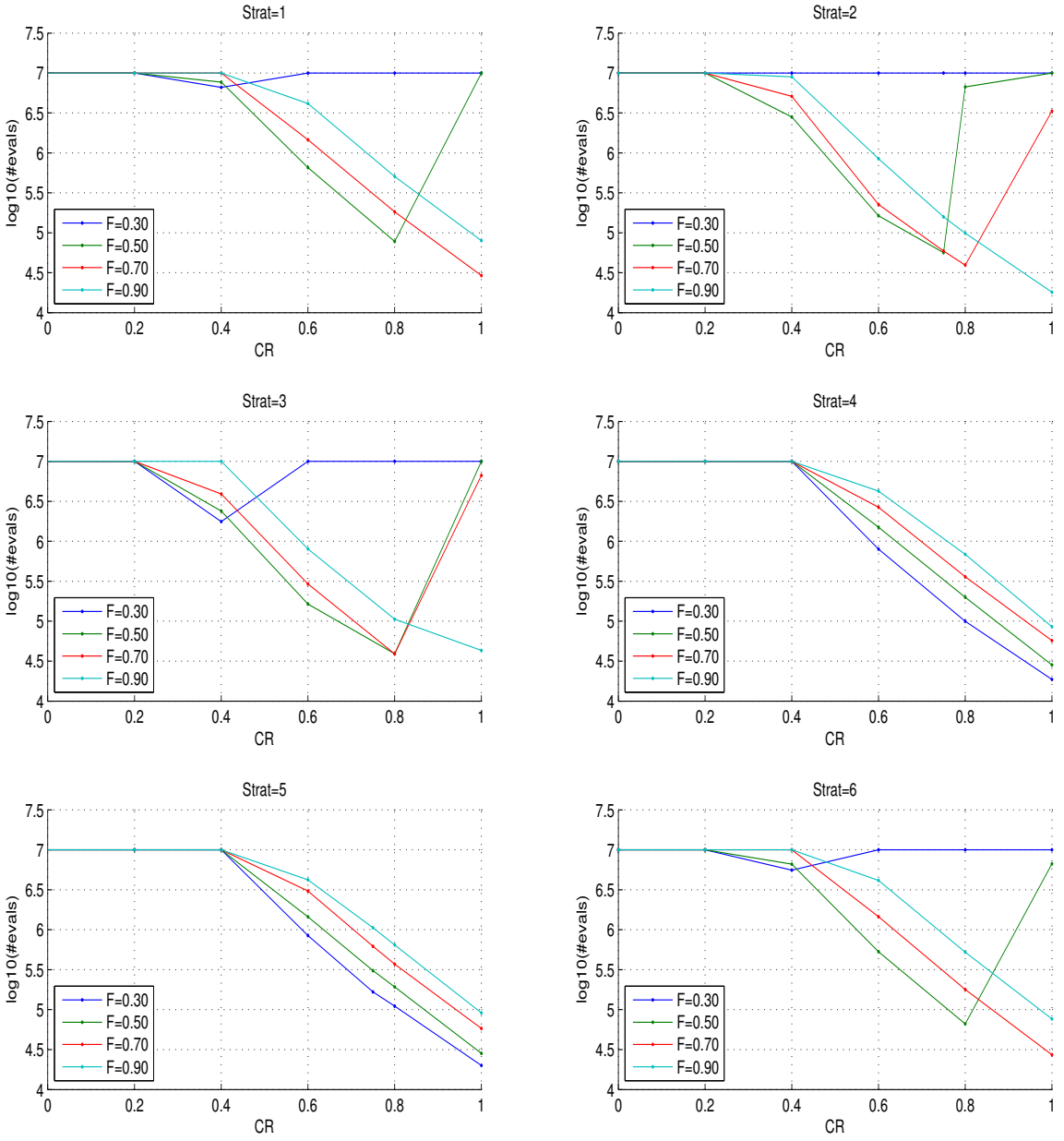


Figure A.8: Parameter identification of DE on the rotated ellipsoid function in 10-D with a population size of five times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

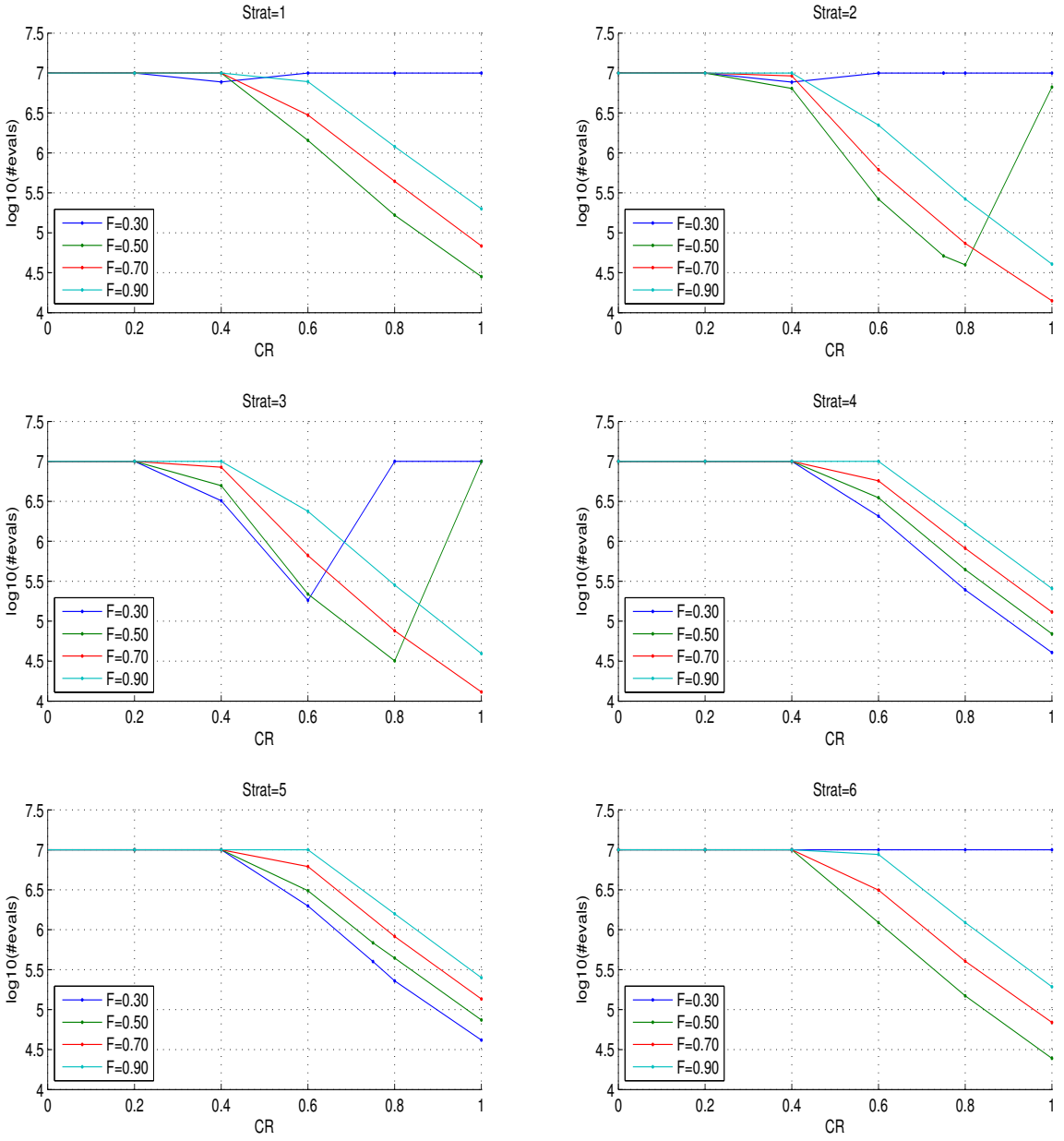


Figure A.9: Parameter identification of DE on the rotated ellipsoid function in 10-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

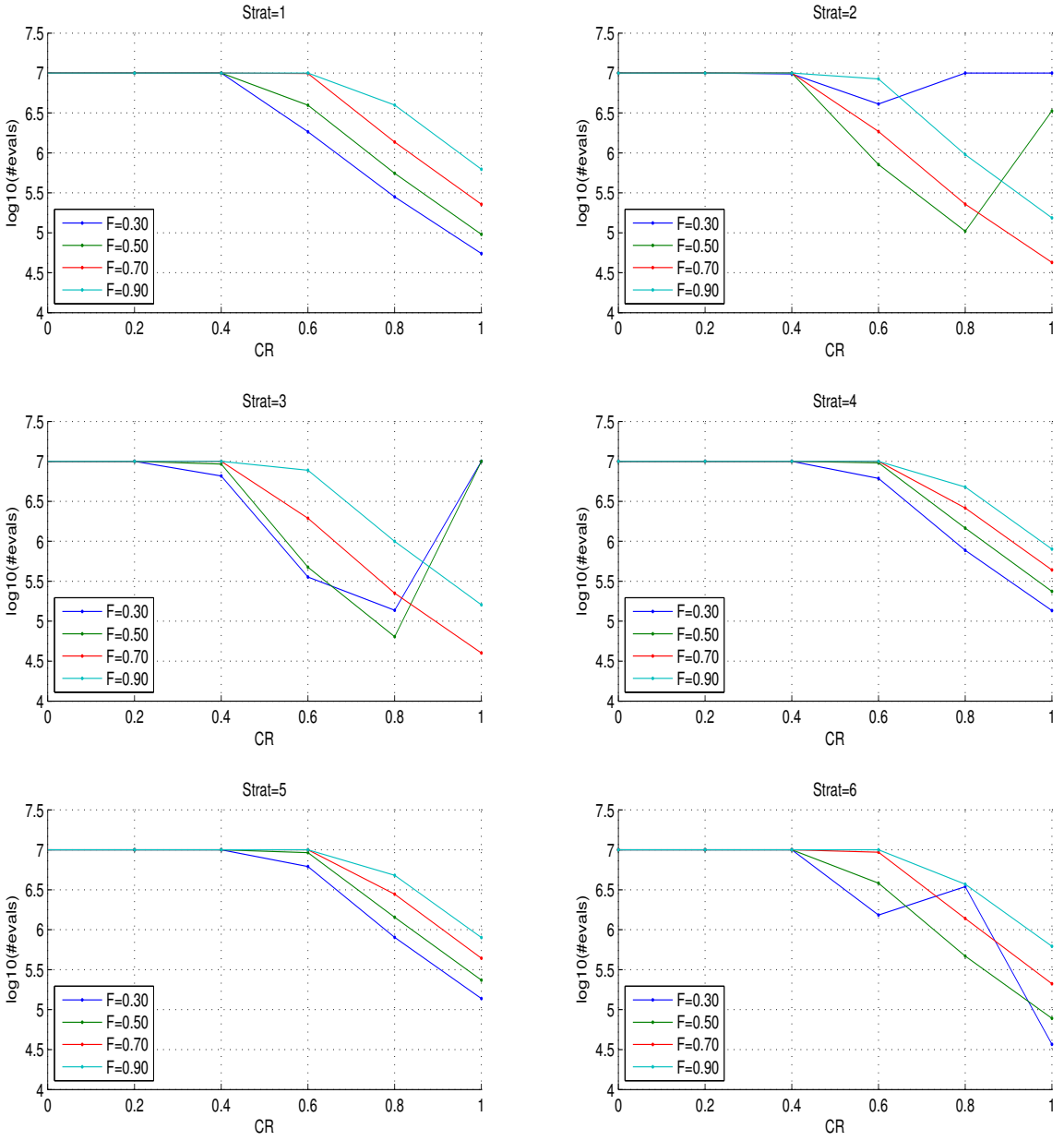


Figure A.10: Parameter identification of DE on the rotated ellipsoid function in 10-D with a population size of thirty times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

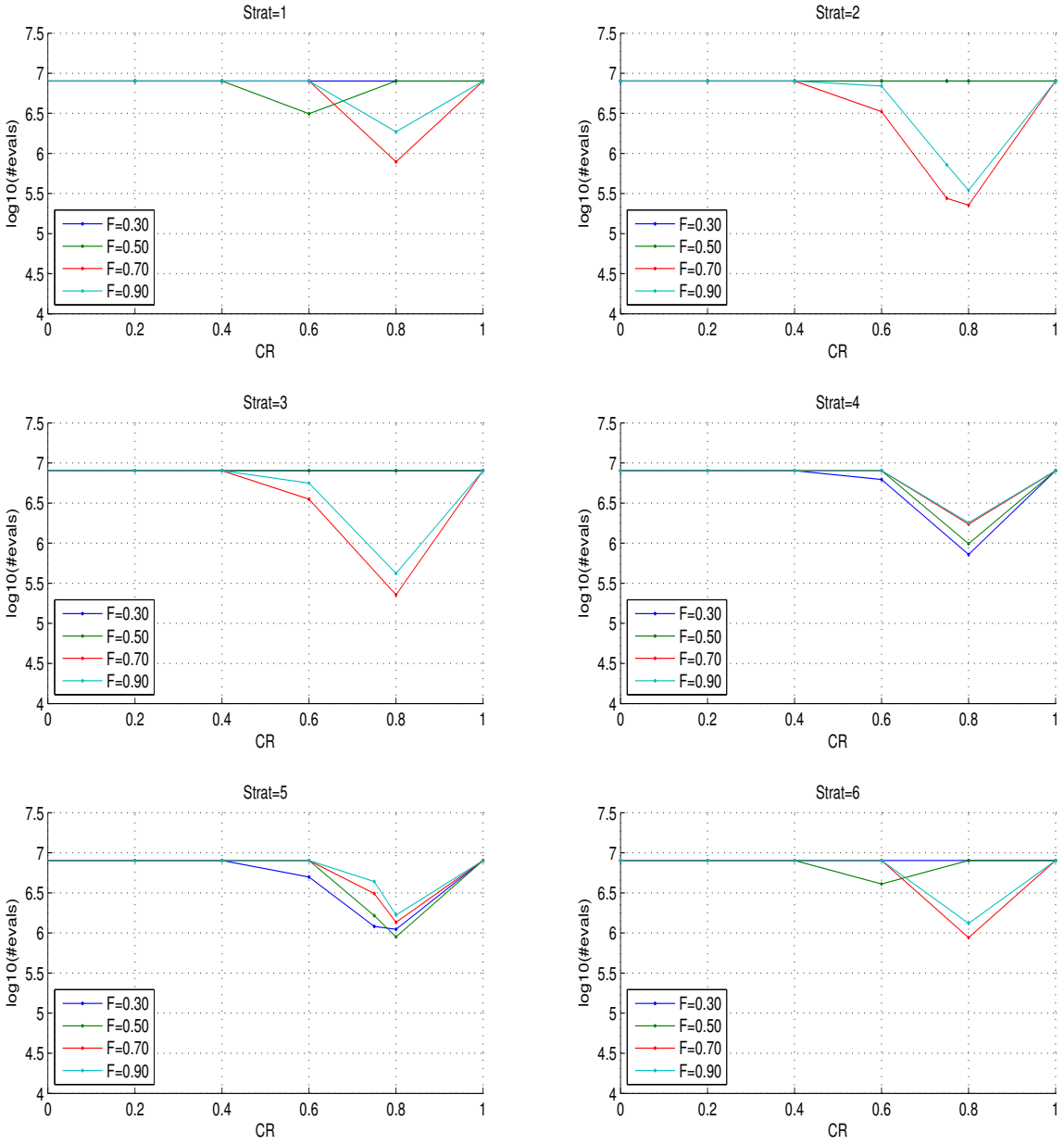


Figure A.11: Parameter identification of DE on the rotated ellipsoid function in 20-D with a population size of one times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

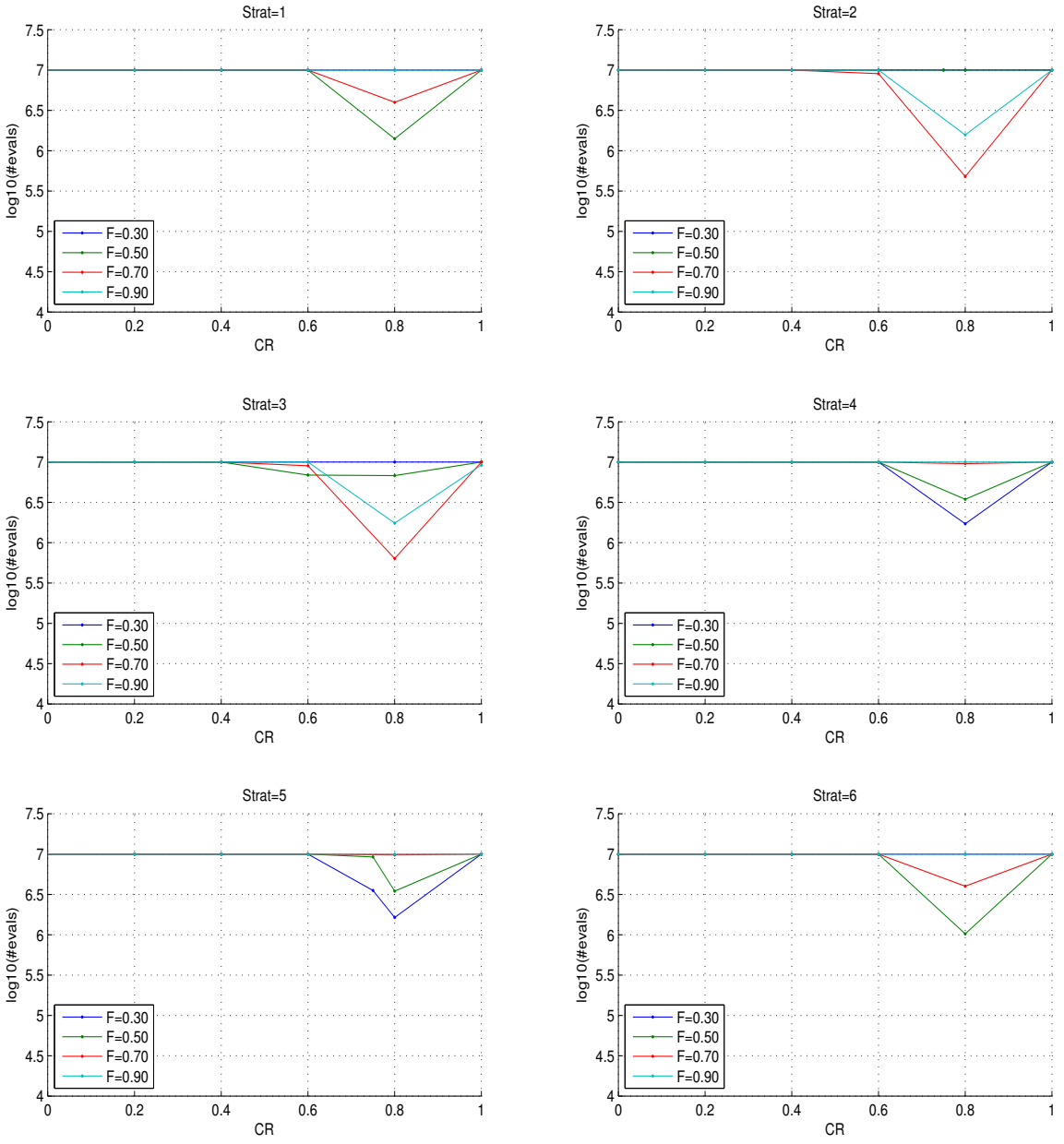


Figure A.12: Parameter identification of DE on the rotated ellipsoid function in 20-D with a population size of three times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

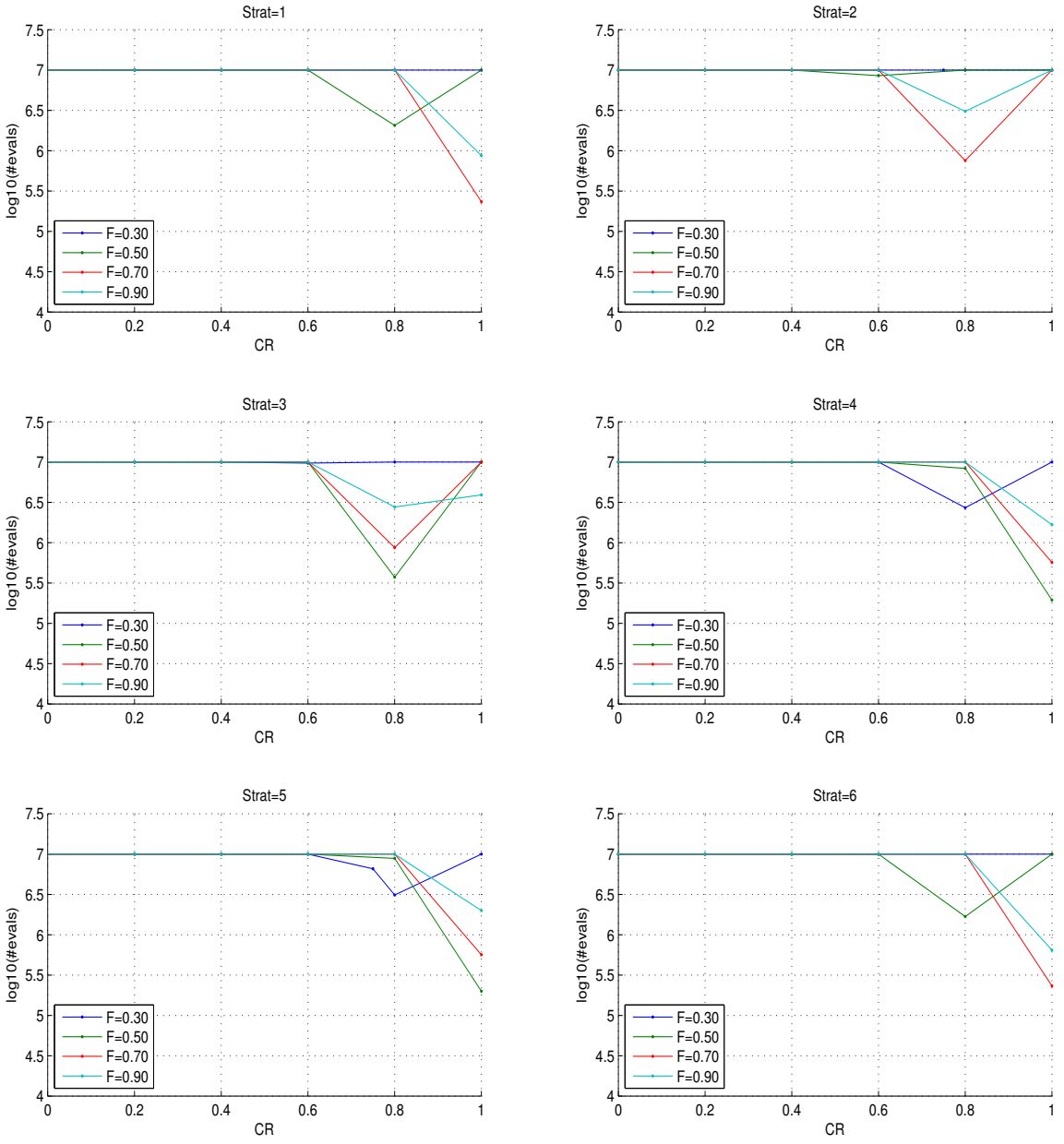


Figure A.13: Parameter identification of DE on the rotated ellipsoid function in 20-D with a population size of five times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

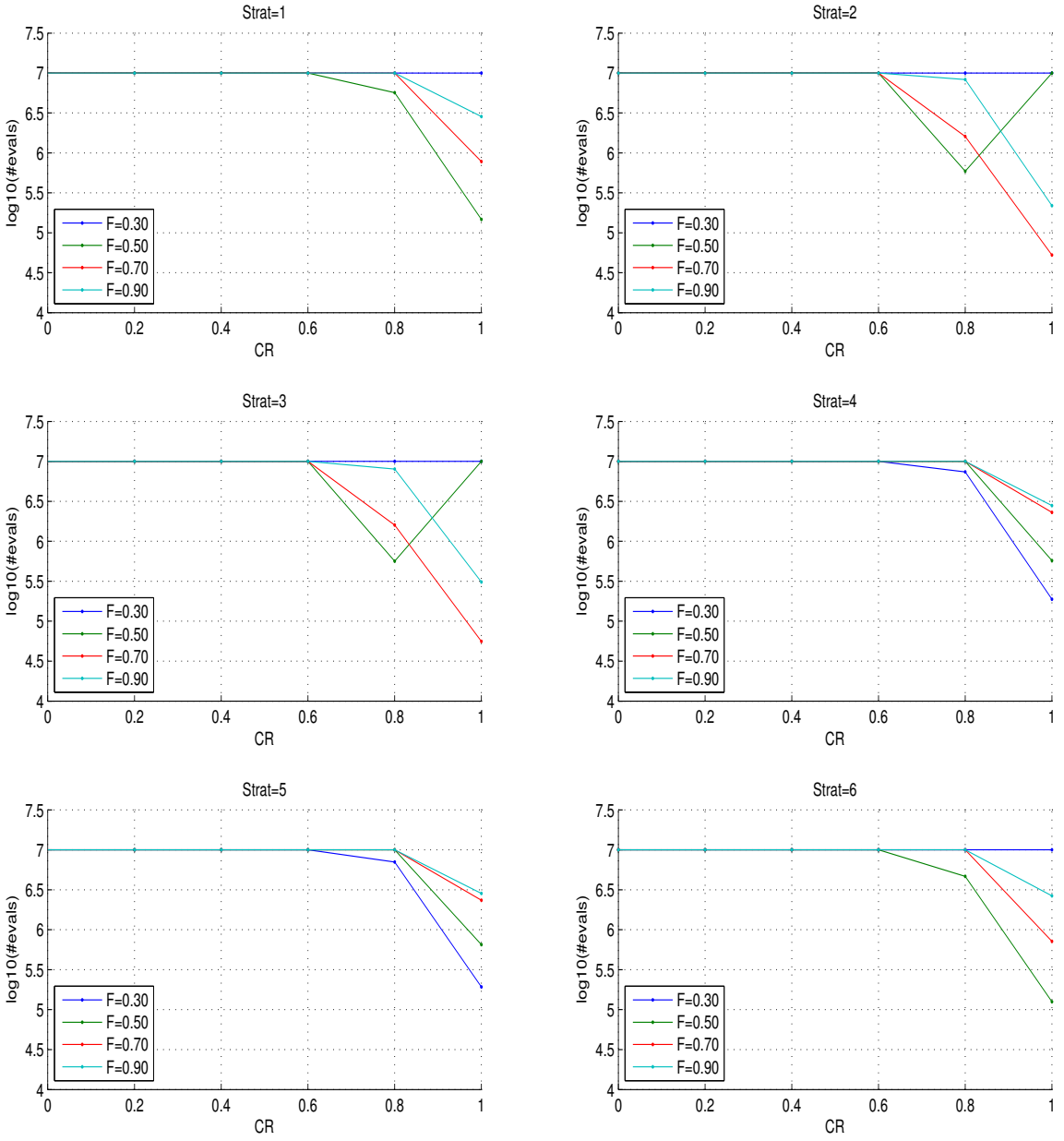


Figure A.14: Parameter identification of DE on the rotated ellipsoid function in 20-D with a population size of ten times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

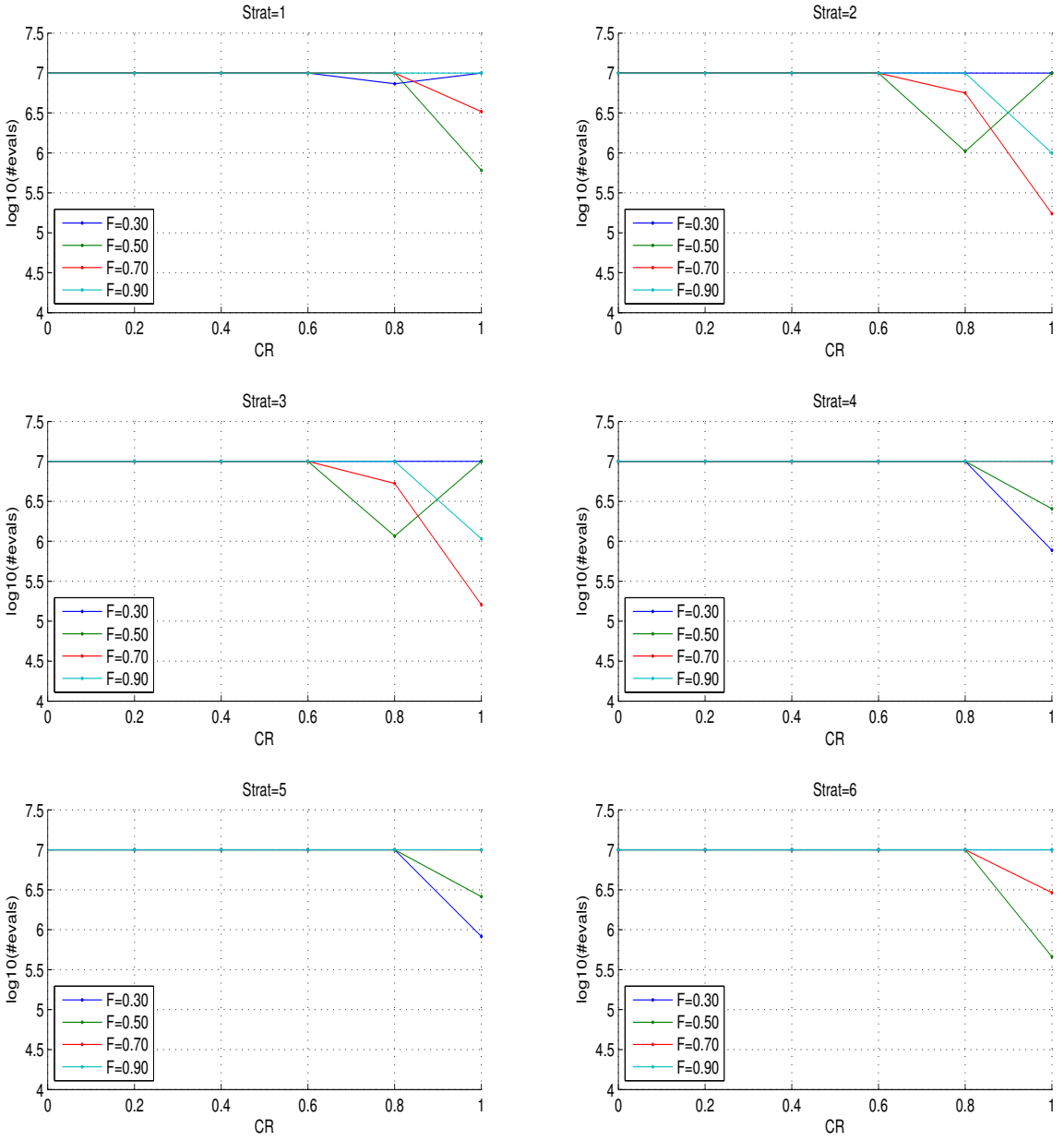


Figure A.15: Parameter identification of DE on the rotated ellipsoid function in 20-D with a population size of thirty times the dimension. The logarithm in base 10 of the average number of function evaluations to reach the target function value 10^{-7} is shown for different values of CR and F. The six sub-figures from left to right, top to bottom correspond to the variants: 1. DE/rand/1, 2. DE/local-to-best/1, 3. DE/best/1 with jitter, 4. DE/rand/1 with per-vector-dither, 5. DE/rand/1 with per-generation-dither and 6. DE/rand/1 either-or successively.

When the population is smaller than five times the dimension, the algorithm does not converge as fast. When the population size is larger than ten times the dimension, the associated search costs are too prohibitive.

These results led us to consider the DE/local-to-best/1/bin variant, also denoted as DE/target-to-best/1/bin in [Price et al., 2005] which uses a single difference between a random vector and the best-so-far vector and uniform cross-over with $F = 0.8$ and $CR = 1$ and default population size of ten times the dimension.

Appendix B

Empirical Cumulative Distribution Functions of Empirical Running Time

The ERT, expected running time, to reach a target function value is the number of function evaluations over all runs divided by the number of runs that surpassed the considered function value. The ERT is the most prominent performance measure used in BBOB 2009. We describe here our use of the Empirical Cumulative Distribution Functions, ECDFs, of the bootstrap distribution of the ERT divided by the problem dimension n. Especially we discuss the relation between these ECDFs and the traditional convergence graphs that present the data of runs as function values reached versus time. More specifically, we consider the monotonic convergence graphs that present the best function values obtained versus time for an optimisation run. The ERT are divided by the dimension so the values are more easily compared for different dimensions.

B.1 Horizontal Versus Vertical View

As for representations in terms of BBO benchmarking, there are two paradigms that the practitioner can consider:

the fixed-cost scenario, which is denoted as vertical view since in a convergence

graph (function values versus time) fixing a cost is putting a straight vertical line and considering the intersection of the graphs with it;

the fixed-target scenario, which is denoted as horizontal view since likewise it corresponds to a horizontal straight line in a convergence graph.

The idea of fixed-cost is anchored in everyday reality: 'given a budget what is the best result one can obtain?'. In the context of continuous optimisation, this means 'what is the best function value an algorithm can achieve'. The algorithms can then be ranked by function values reached. The horizontal view, or fixed-target scenario results in a ranking of algorithms by costs and also quantitative comparison of the form: 'algorithm $\mathcal{A}$ is faster than algorithm $\mathcal{B}$ by a factor of X on a given test function'. The topic of the horizontal versus vertical view is discussed in more details in [Hansen et al., 2009a].

B.2 Explanation of Empirical Cumulative Distribution Functions

Our goal is to obtain a condensed representation of the convergence graphs figure. Given a straight line cut into the space of the convergence graphs, the intersections of the trajectories in the convergence graph provide us with an empirical distribution, see Figure B.1. Suppose the cut is horizontal, meaning we are considering a fixed-target, we would obtain the empirical distribution of the costs needed to attain such target. The aforementioned empirical distribution can be represented as an ECDF which estimates the probability of the considered event to occur; in the case of the horizontal cut, it is the probability that the associated target is attained for a given cost, see Figure B.2. If we now consider two parallel cuts, the resulting ECDF will estimate the probability that any of the two events occur. Let us consider many horizontal cuts in the space of the convergence graphs of a single algorithm, the resulting ECDF would represent the probability that any such targets is attained for a given cost, see Figure B.2. In the case of vertical cuts *i.e.* many fixed-budgets, the resulting ECDF would represent the probability that any of these budgets is sufficient to reach a given function value.

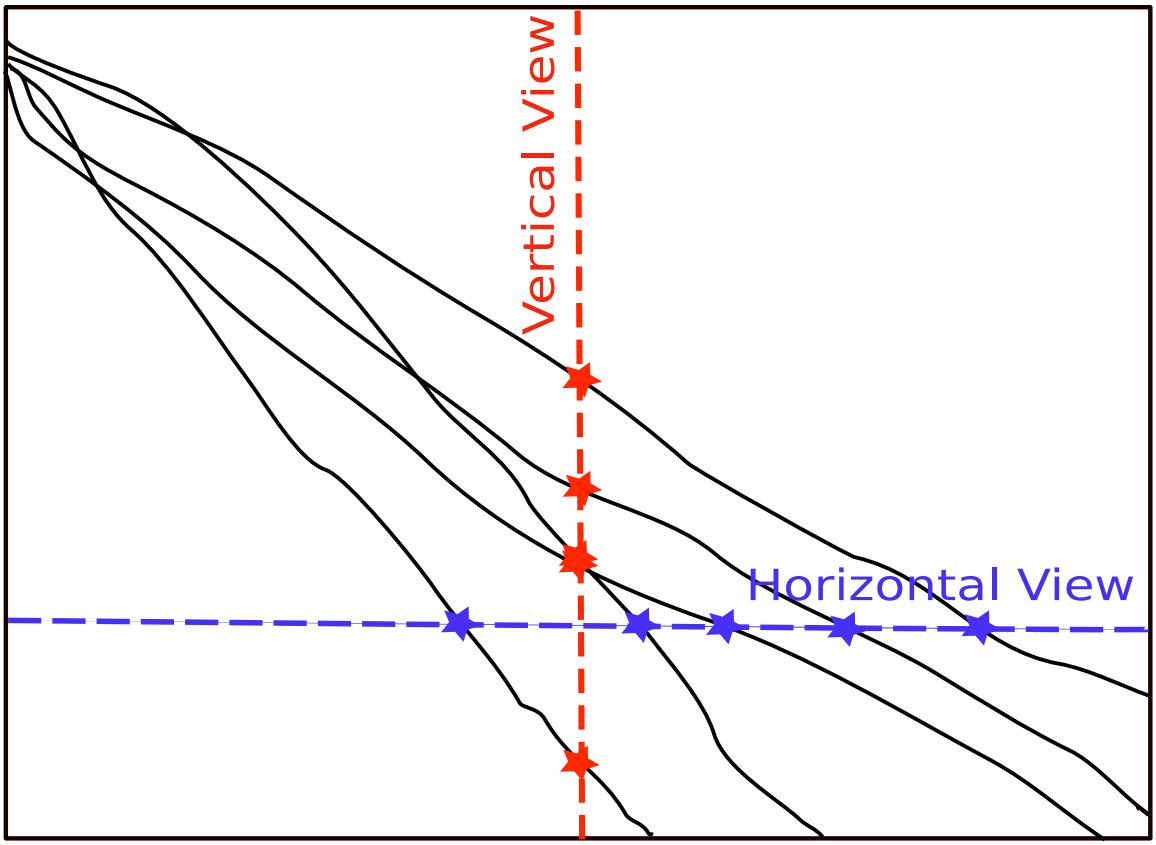


Figure B.1: Horizontal view or fixed-target scenario and vertical view or fixed-budget scenario for convergence graphs, the axis represent function values versus time [Hansen et al., 2009a]

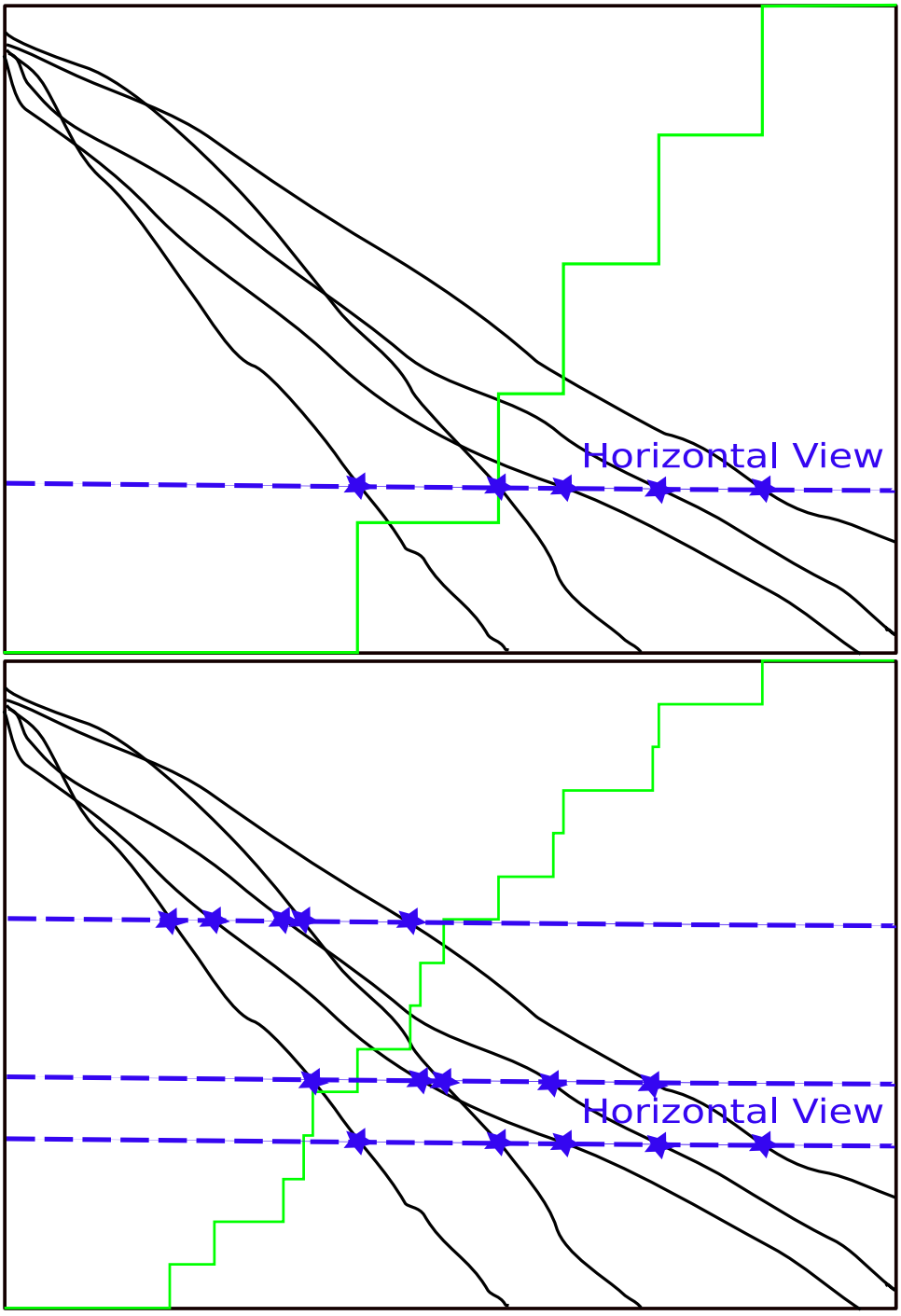


Figure B.2: Empirical Cumulative Distribution Function for a given fixed-target (top sub-figure), for three such fixed-targets (bottom), the y-axis range of the empirical cumulative distribution function is $[0, 1]$ and differs from that of the convergence graphs

So far, we have made no assumption on the number of problems considered. The convergence graphs considered might be obtained from different objective functions. Thus, the ECDFs can not only be used to aggregate the information over multiple runs, for instance on a single objective function, but also the information over many different problems.

B.3 Uniform Targets versus Variable Targets

Let us consider convergence graphs. There is a high chance that the easiest targets, which are the horizontal cuts closer to the top of convergence graphs, are attained first, and that the hardest targets —cuts at the bottom— are attained last. The hardest targets are likely to be under-represented in the ECDFs, see Figure B.2, because optimisation runs might stop before attaining such targets. Distributing targets on a log scale allows to have more cuts in the region of the hardest targets.

The issue of distributing targets is obvious if we consider the convergence graphs on different objective functions: as discussed in Section 4.4.4 a given target function value may correspond to different levels of difficulty on different objective functions in terms of computational costs. Thus, we choose the targets depending on the computational costs involved for the algorithms considered.

Choosing the targets depending on the costs for an algorithm is done by considering different costs —vertical cuts—, and a reference algorithm $\mathcal{A}$. The intersections of the trajectories of $\mathcal{A}$ with the vertical cuts give function values associated to given fixed-costs $f(\mathcal{A}, \text{running time})$. These resulting function values give us a distribution of horizontal cuts related to the performances of reference algorithm $\mathcal{A}$ which can be used to generate ECDFs, see Figure B.3. In the example given in the Figure B.3, the reference algorithm is the second best for a given budget.

B.4 Bootstrapping

We explain here how an ECDF of ERT can be extended to the right with bootstrapping.

Bootstrapping [Efron and Tibshirani, 1993] was used in BBOB 2009 to give a

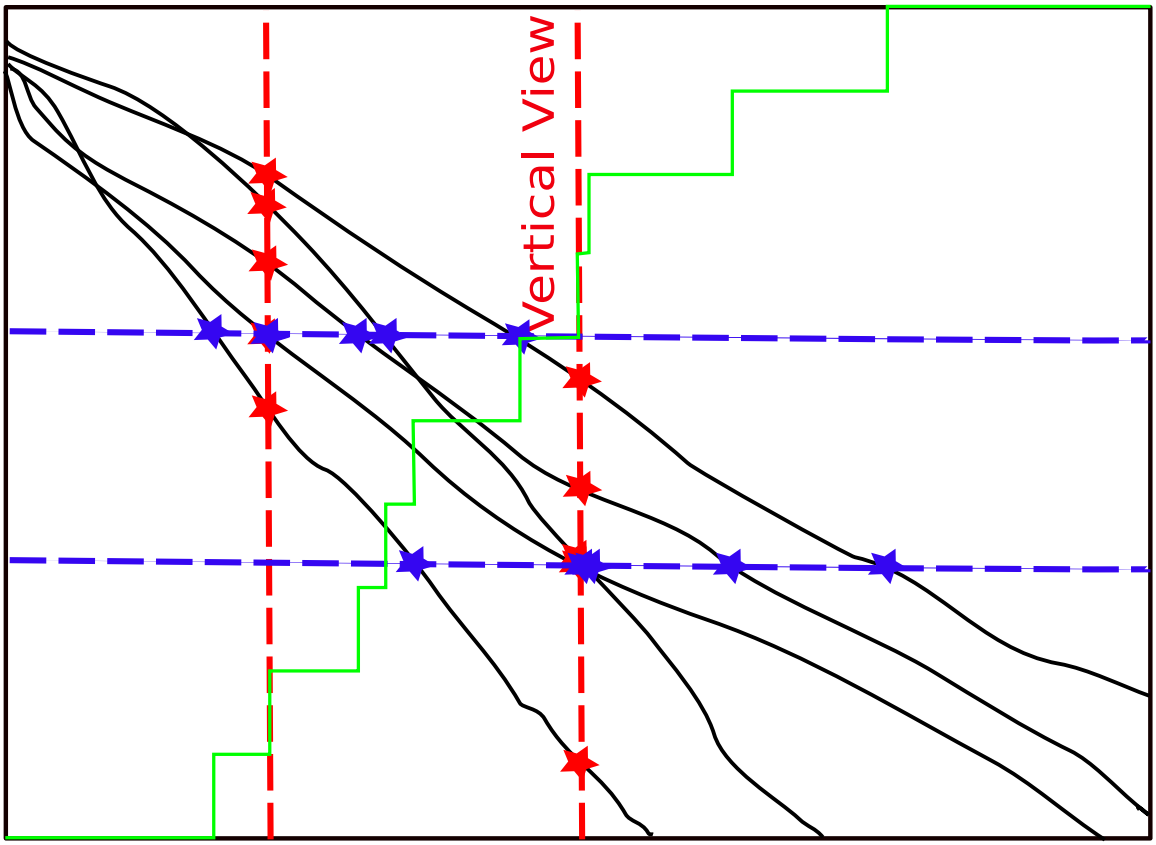


Figure B.3: Empirical Cumulative Distribution Function for two fixed-targets, the corresponding horizontal cuts are provided by the second best trajectory for given budgets (vertical cuts)

dispersion measure for ERT: the ERT gives a single measurement from a data sample —from `Ntrials` optimisation runs in BBOB 2009— and bootstrapping provide samples by drawing from the original data sample with replacement [Hansen et al., 2009a]. The ECDFs of ERT for data samples are bounded to the right by the maximum total number of function evaluations in these data samples. The ECDFs of the bootstrap distribution of ERT are not bounded to the right because sampling the bootstrap distribution of ERT might generate values larger than the maximum total number of function evaluations.

B.5 Comparisons with other representations

The ECDFs of the ERT to reach a target function value are comparable to the data profiles and the performance profiles proposed in [Moré and Wild, 2009]. The data profiles are empirical cumulative distribution functions of running lengths to attain a relative precision, divided by twice the dimension plus one function evaluations which is the size of a simplex. The performance profiles are empirical cumulative distribution functions of running lengths to attain a relative precision normalised by those of the best algorithm.

Both data profiles and performance profiles provide complementary views of the performances of an algorithm that can be retrieved when considering both uniform and variable targets for our empirical cumulative distribution functions of ERT. Data profiles can clearly be assimilated to the empirical cumulative distribution functions of ERT divided by the dimension using uniform target function values (horizontal view), whereas the normalisation used in performance profiles is comparable to the variable targets strategy which also use ranking of algorithms.

Appendix C

Installing bbob_pproc

The entire post-processing tool is written in PYTHON and requires Python to be installed on your machine. The minimal software requirements for using the post-processing tool are Python (2.5.2), Matplotlib (0.91.2) and Numpy (1.0.4). In the following, we explain how to obtain and install the required software for different systems (Linux, Windows, Mac OS) and which steps you have to perform to run the post-processing on your data.

While the `bbob_pproc` source files are provided, you need to install Python and its libraries Matplotlib and Numpy. We recommend using Python 2.5 and not a higher version (2.6 or 3.0) since the necessary libraries are not (yet) available and the code is not verified.

C.1 Downloading the Packages

For all operating systems the packages can be found at the following locations:

- Python: http://www.python.org/download/releases/2.5.4/,
- Numpy: http://sourceforge.net/projects/numpy/,
- Matplotlib: http://sourceforge.net/projects/matplotlib/.

We recommend the use of the latest versions of Matplotlib (0.98.5.2) and Numpy (1.2.1).

C.2 Installing on Linux

In most common Linux distributions Python (but not Numpy or Matplotlib) is already part of the installation. If not, use your favourite package manager to install Python (package name: python), Numpy (python-numpy) and Matplotlib (package name: python-matplotlib) and their dependencies. If your distribution and repositories are up-to-date, you should have at least Python 2.5.2, Matplotlib 0.91.2 and Numpy 1.0.4. Though those are not the most recent versions of each package, they meet the minimal software requirements to make the BBOB software work. If needed, you can alternatively download sources and compile binaries. Python and the latest versions of Matplotlib and Numpy can be downloaded from the links in Section C.1. A dependency for the Linux version of Matplotlib is libpng, which can be obtained at http://www.libpng.org/. You then need to properly install the downloaded packages before you can use them. Please refer to the corresponding package installation pages.

C.3 Installing on Windows

For installing Python under Windows, please go to the Python link in Section C.1 and download `python-2.5.4.msi`. This file requires the Microsoft Installer, which is a part of Windows XP and later releases. If you don't have the Microsoft Installer, there is a link for the download provided at the same page. After installing Python, it is recommended to first install Numpy and then Matplotlib. Both can be installed with the standard .exe files which are respectively

- `numpy-1.2.1-win32-superpack-python2.5.exe` and,
- `matplotlib-0.98.5.2.win32-py2.5.exe`.

These files can be obtained from the provided SourceForge links in Section C.1.

C.4 Installing on Mac OS

Mac OS X comes with Python pre-installed, the version might be older than 2.5 though. It is recommended to upgrade Python by downloading and installing a newer

version. To do this, if you have Mac OS X 10.3 and later you can download the disk image file `python-2.5.4-macosx.dmg` containing universal binaries from the Python download page, see Section C.1. More information on the update of Python on Mac OS can be found at this location: http://www.python.org/download/mac/[1]. Open the disk image and use the installer[2]. You then need to download and install Numpy and Matplotlib from the SourceForge links listed in Sect C.1.

[1]The discussion over IDLE for Leopard user (http://wiki.python.org/moin/MacPython/Leopard) is not relevant for the use of bbob_pproc package.

[2]Following this step leave the pre-installed Python on the system and install the MacPython 2.5.4 distribution. MacPython contains a Python installation as well as some Mac-specific extras.

Bibliography

Angelo Marcello Anile, Vincenzo Cutello, Giuseppe Nicosia, Rosario Rascunà, and Salvatore Spinella. Comparison among evolutionary algorithms and classical optimization methods for circuit design problems. In *Congress on Evolutionary Computation*, pages 765–772. IEEE, 2005. ISBN 0-7803-9363-5.

D. Arnold and R. Salomon. Evolutionary gradient search revisited. *IEEE Transactions on Evolutionary Computation*, 11(4):480–495, 2007.

A Auger and N Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776. IEEE Press, 2005.

A. Auger, N. Hansen, J. M. Perez Zerpa, R. Ros, and M. Schoenauer. Empirical comparisons of several derivative free optimization algorithms. In *Acte du 9ieme colloque national en calcul des structures*, May 2009a.

Anne Auger. Benchmarking the (1+1) evolution strategy with one-fifth success rule on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2447–2452. ISBN 978-1-60558-505-5.

Anne Auger. Benchmarking the (1+1)-ES with one-fifth success rule on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2453–2458. ISBN 978-1-60558-505-5.

Anne Auger and Nikolaus Hansen. Benchmarking the (1+1)-CMA-ES on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2459–2466. ISBN 978-1-60558-505-5.

Anne Auger and Nikolaus Hansen. Benchmarking the (1+1)-CMA-ES on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2467–2472. ISBN 978-1-60558-505-5.

Anne Auger and Raymond Ros. Benchmarking the pure random search on the BBOB-2009 testbed. In Rothlauf [2009], pages 2479–2484. ISBN 978-1-60558-505-5.

Anne Auger and Raymond Ros. Benchmarking the pure random search on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2485–2490. ISBN 978-1-60558-505-5.

Anne Auger, Nikolaus Hansen, J. M. Perez Zerpa, Raymond Ros, and Marc Schoenauer. Experimental comparisons of derivative free optimization algorithms. In Jan Vahrenhold, editor, *SEA*, volume 5526 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2009b. ISBN 978-3-642-02010-0.

R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

G. Bilchev and I.C. Parmee. Ant colony search vs. genetic algorithms. Technical report, Plymouth Engineering Design Centre, University of Plymouth, 1995.

Peter A. N. Bosman and Dirk Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The idea. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 767–776, London, UK, 2000. Springer-Verlag. ISBN 3-540-41056-2.

Peter A. N. Bosman, Jörn Grahl, and Dirk Thierens. AMaLGaM IDEAs in noiseless black-box optimization benchmarking. In Rothlauf [2009], pages 2247–2254. ISBN 978-1-60558-505-5.

Peter A. N. Bosman, Jörn Grahl, and Dirk Thierens. AMaLGaM IDEAs in noisy black-box optimization benchmarking. In Rothlauf [2009], pages 2351–2358. ISBN 978-1-60558-505-5.

S. H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6:244– 251, 1958.

Jason Brownlee. A note on research methodology and benchmarking optimization algorithms. Technical Report 070125, Complex Intelligent Systems Laboratory

(CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, January 2007a. URL http://www.ict.swin.edu.au/personal/jbrownlee/2007/TR01-2007.pdf.

Jason Brownlee. Oat: The optimization algorithm toolkit. Technical Report 20071220A, Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, December 2007b.

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593, October 1965. URL http://www.jstor.org/stable/2003941.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.

M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, Paris, 1991. MIT Press/Bradford Book.

A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In *Nonlinear Optimization and Applications*, pages 27–47, New York, NY, USA, 1996. Kluwer Academic/Plenum Publishers.

A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Math. Program.*, 79(1-3):397–414, 1997a. ISSN 0025-5610. doi: http://dx.doi.org/10.1007/BF02614326.

A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108. Cambridge University Press, 1997b. URL http://perso.fundp.ac.be/~phtoint/pubs/TR96-10.abstract.

A. R. Conn, K. Scheinberg, and Ph. L. Toint. A derivative free optimization algorithm in practice. In *Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO*. AIAA, 1998.

A. R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-region methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. ISBN 0-89871-460-5.

Tibor Csendes. Nonlinear parameter estimation by global optimization—efficiency and reliability. *Acta Cybern.*, 8(4):361–370, 1989. ISSN 0324-721X.

L. Devroye. The compund random search. In *International Symposium on Systems Engineering and Analysis*, pages 195–110. Purdue University, 1972.

M. A. Diniz-Ehrhardt, J. M. Martínez, and M. Raydan. A derivative-free nonmonotone line-search technique for unconstrained optimization. *J. Comput. Appl. Math.*, 219(2):383–397, 2008. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/j.cam.2007.07.017.

L. C. W. Dixon, J. Gomulka, and S. E. Herson. Reflection on global optimization problems. In *Optimization in Action*, pages 398–435. Academic Press, New York, 1976.

Benjamin Doerr, Mahmoud Fouz, Martin Schmidt, and Magnus Wahlström. BBOB: Nelder-mead with resize and halfruns. In Rothlauf [2009], pages 2239–2246. ISBN 978-1-60558-505-5.

B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Chapman & Hall/CRC, 1993.

Mohammed El-Abd and Mohamed S. Kamel. Black-box optimization benchmarking for noiseless function testbed using an EDA and PSO hybrid. In Rothlauf [2009], pages 2263–2268. ISBN 978-1-60558-505-5.

Mohammed El-Abd and Mohamed S. Kamel. Black-box optimization benchmarking for noiseless function testbed using particle swarm optimization. In Rothlauf [2009], pages 2269–2274. ISBN 978-1-60558-505-5.

Mohammed El-Abd and Mohamed S. Kamel. Black-box optimization benchmarking for noiseless function testbed using PSO_Bounds. In Rothlauf [2009], pages 2275–2280. ISBN 978-1-60558-505-5.

Vitaliy Feoktistov. *Differential Evolution: In Search of Solutions*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387368957.

S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, 2009a.

S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noisy functions. Technical Report 2009/21, Research Center PPE, 2009b.

R. Fletcher. A new approach to variable metric algorithms. *Computer journal*, 13: 317–322, 1970.

R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, New York, NY, USA, 1987. ISBN 0-471-91547-5.

K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis Jr., C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, May 2008. ISSN 0309-1708. doi: 10.1016/j.advwatres.2008.01.010. URL http://www.sciencedirect.com/science/article/B6VCF-4RRFNH4-2/2/4811ea6adb8743866ec24fe8e0e09d45.

Marcus Gallagher. Black-box optimization benchmarking: results for the BayEDAcG algorithm on the noiseless function testbed. In Rothlauf [2009], pages 2281–2286. ISBN 978-1-60558-505-5.

Marcus R. Gallagher. Black-box optimization benchmarking: results for the BayEDAcG algorithm on the noisy function testbed. In Rothlauf [2009], pages 2383–2388. ISBN 978-1-60558-505-5.

Carlos García-Martínez and Manuel Lozano. A continuous variable neighbourhood search based on specialised EAs: application to the noiseless BBO-benchmark 2009. In Rothlauf [2009], pages 2287–2294. ISBN 978-1-60558-505-5.

Carlos García-Martínez and Manuel Lozano. A continuous variable neighbourhood search based on specialised EAs: application to the noisy BBO-benchmark 2009 testbed. In Rothlauf [2009], pages 2367–2374. ISBN 978-1-60558-505-5.

José García-Nieto, Enrique Alba, and Javier Apolloni. Noiseless functions black-box optimization: evaluation of a hybrid particle swarm with differential operators. In Rothlauf [2009], pages 2231–2238. ISBN 978-1-60558-505-5.

José García-Nieto, Enrique Alba, and Javier Apolloni. Particle swarm hybridized with differential evolution: black box optimization benchmarking for noisy functions. In Rothlauf [2009], pages 2343–2350. ISBN 978-1-60558-505-5.

P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM J. Optim*, 5:269–285, 1995.

D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. Cuter and sifdec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Softw.*, 29(4):373–394, 2003. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/962437.962439.

L Grippo, F Lampariello, and S Lucidi. A nonmonotone line search technique for newton's method. *SIAM J. Numer. Anal.*, 23(4):707–716, 1986. ISSN 0036-1429. doi: http://dx.doi.org/10.1137/0723046.

N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006a.

N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In Xin Yao et al., editors, *Parallel Problem Solving from Nature - PPSN VIII, LNCS 3242*, pages 282–291. Springer, 2004.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

N. Hansen, A. Ostermeier, and A. Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In L. J. Eshelman, editor, *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, 1995.

N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation. *Evolutionary Computation*, 11(1):1–18, 2003.

N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, 2009a. URL http://hal.inria.fr/inria-00362649/en/.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009b. URL http://hal.inria.fr/inria-00362633/en/.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions. Technical Report RR-6869, INRIA, 2009c. URL http://hal.inria.fr/inria-00369466/en.

Nikolaus Hansen. Compilation of results on the 2005 CEC benchmark function set. Online, May 2006b. URL http://www.bionik.tu-berlin.de/user/niko/cec2005compareresults.pdf.

Nikolaus Hansen. Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2389–2396. ISBN 978-1-60558-505-5.

Nikolaus Hansen. Benchmarking a bi-population CMA-ES on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2397–2402. ISBN 978-1-60558-505-5.

Nikolaus Hansen. Benchmarking the nelder-mead downhill simplex algorithm with many local restarts. In Rothlauf [2009], pages 2403–2408. ISBN 978-1-60558-505-5.

Nikolaus Hansen. The CMA-ES evolution strategy: A tutorial. Online, January 2009d. URL http://www.lri.fr/~hansen/cmatutorial.pdf.

Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. Impacts of invariance in search: when CMA-ES and PSO face ill-conditioned problems. To be published, March 2009d. URL http://www.lri.fr/~hansen/psopaper09.pdf.

J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

Robert Hooke and T. A. Jeeves. "Direct Search" solution of numerical and statistical problems. *J. ACM*, 8(2):212–229, 1961.

Reiner Horst. *Introduction to Global Optimization (Nonconvex Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 0792335562.

Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *J. of Global Optimization*, 14(4):331–355, 1999. ISSN 0925-5001. doi: http://dx.doi.org/10.1023/A:1008382309369.

Waltraud Huyer and Arnold Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM Trans. Math. Softw.*, 35(2):1–25, 2008. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/1377612.1377613.

Waltraud Huyer and Arnold Neumaier. Benchmarking of SNOBFIT on the noisy function testbed. Online, 2009a. URL http://www.mat.univie.ac.at/~neum/papers.html. P. 987.

Waltraud Huyer and Arnold Neumaier. Benchmarking of MCS on the noisy function testbed. Online, 2009b. URL http://www.mat.univie.ac.at/~neum/papers.html. P. 988.

Waltraud Huyer and Arnold Neumaier. Benchmarking of MCS on the noiseless function testbed. Online, 2009c. URL http://www.mat.univie.ac.at/~neum/papers.html. P. 989.

Christian Igel, Thorsten Suttorp, and Nikolaus Hansen. A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 453–460, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: http://doi.acm.org/10.1145/1143997.1144082.

D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 of *DIMACS*, pages 215—250. American Mathematical Society, 2002.

Steven G. Johnson. The nlopt nonlinear-optimization package. WWW, 2008. URL http://ab-initio.mit.edu/nlopt.

D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *J. Optim. Theory Appl.*, 79(1):157–181, 1993. ISSN 0022-3239. doi: http://dx.doi.org/10.1007/BF00941892.

C. T. Kelley. *Iterative Methods for Optimization*. SIAM, 1999.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Con ference on*, volume 4, pages 1942–1948, 1995.

S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

James N. Knight and Monte Lunacek. Reducing the space-time complexity of the CMA-ES. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 658–665, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: http://doi.acm.org/10.1145/1276958.1277097.

Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45:385–482, 2003.

Peter Korosec and Jurij Silc. A stigmergy-based algorithm for black-box optimization: noiseless function testbed. In Rothlauf [2009], pages 2295–2302. ISBN 978-1-60558-505-5.

Peter Korosec and Jurij Silc. A stigmergy-based algorithm for black-box optimization: noisy function testbed. In Rothlauf [2009], pages 2375–2382. ISBN 978-1-60558-505-5.

Jirí Kubalik. Black-box optimization benchmarking of prototype optimization with evolved improvement steps for noiseless function testbed. In Rothlauf [2009], pages 2303–2308. ISBN 978-1-60558-505-5.

Sergei Kucherenko. Application of quasi monte carlo methods in global optimization. In *Global Optimization*, volume 84 of *Nonconvex Optimization and Its Applications*, pages 111–133. Springer US, 2006. doi: 10.1007/0-387-30528-9.

Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM J. on Optimization*, 9(1):112–147, 1998. ISSN 1052-6234. doi: http://dx.doi.org/10.1137/S1052623496303470.

W.B. Langdon and R. Poli. Evolving problems to learn about particle swarm optimizers and other search algorithms. *Evolutionary Computation, IEEE Transactions on*, 11(5):561–578, 2007. ISSN 1089-778X. doi: {10.1109/TEVC.2006.886448}.

Pedro Larrañaga and José A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 0792374665.

R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM J. Res. Dev.*, 47(1):57–66, 2003. ISSN 0018-8646.

Monte Lunacek, Darrell Whitley, and Andrew Sutton. The impact of global structure on search. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pages 498–507, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87699-1. doi: http://dx.doi.org/10.1007/978-3-540-87700-4_50.

K. I. M. McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM J. on Optimization*, 9(1):148–158, 1998. ISSN 1052-6234. doi: http://dx.doi.org/10.1137/S1052623496303482.

Daniel Molina, Manuel Lozano, and Francisco Herrera. A memetic algorithm using local search chaining for black-box optimization benchmarking 2009 for noise free functions. In Rothlauf [2009], pages 2255–2262. ISBN 978-1-60558-505-5.

Daniel Molina, Manuel Lozano, and Francisco Herrera. A memetic algorithm using local search chaining for black-box optimization benchmarking 2009 for noisy functions. In Rothlauf [2009], pages 2359–2366. ISBN 978-1-60558-505-5.

Jorge J. Moré and Stefan M. Wild. Benchmarking derivative-free optimization algorithms. Technical Report ANL/MCS-P1471-1207, Argonne National Laboratory, April 2008.

Jorge J. Moré and Stefan M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optimization*, 20(1):172–191, 2009. doi: 10.1137/080724083.

Heinz Mühlenbein and Gerhard Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 178–187, London, UK, 1996. Springer-Verlag. ISBN 3-540-61723-X.

C. L. Müller and I. F. Sbalzarini. A tunable real-world multi-funnel benchmark problem for evolutionary optimization (and why parallel island models might remedy the failure of CMA-ES on it). In *Proc. Intl. Conf. Evolutionary Computation (ICEC)*, Madeira, Portugal, October 2009. To be published. URL http://www.mosaic.ethz.ch/research/pubs/docs/Muller2009c.pdf.

J.A. Nelder and R. Mead. The downhill simplex method. *Computer Journal*, 7: 308–313, 1965.

George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988. ISBN 0-471-82819-X.

Miguel Nicolau. Application of a simple binary genetic algorithm to a noiseless testbed benchmark. In Rothlauf [2009], pages 2473–2478. ISBN 978-1-60558-505-5.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, 2nd edition, 2006.

A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size Adaptation Based on Non-local Use of Selection Information. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, pages 189–198. Springer-Verlag, LNCS 866, 1994.

László Pál, Tibor Csendes, Mihály Csaba Markót, and Arnold Neumaier. BBO-benchmarking of the GLOBAL method for the noisy function testbed. Online, 2009a. URL http://www.mat.univie.ac.at/~neum/papers.html. P. 985.

László Pál, Tibor Csendes, Mihály Csaba Markót, and Arnold Neumaier. BBO-benchmarking of the GLOBAL method for the noiseless function testbed. Online, 2009b. URL http://www.mat.univie.ac.at/~neum/papers.html. P. 986.

Miroslav Šnorek Pavel Kordík, Oleg Kovářík. Optimization of models: Looking for the best strategy. In *Proceedings of 6th EUROSIM Congress on Modelling and Simulation*, Ljubjana, 2007. ISBN 3-901608-32-X.

Martin Pelikan, David E. Goldberg, and Fernando G. Lobo. A survey of optimization by building and using probabilistic models. *Comput. Optim. Appl.*, 21(1):5–20, 2002. ISSN 0926-6003. doi: http://dx.doi.org/10.1023/A:1013500812258.

János D. Pintér. *Global Optimization in Action, Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications*, volume 6 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht / Boston / London, 1996.

Jan Poland and Andreas Zell. Main vector adaptation: A CMA variant with linear time and space complexity. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1050–1055, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann. ISBN 1-55860-774-9. URL http://citeseer.ist.psu.edu/poland01main.html.

Petr Posik. BBOB-benchmarking a simple estimation of distribution algorithm with Cauchy distribution. In Rothlauf [2009], pages 2309–2314. ISBN 978-1-60558-505-5.

Petr Posik. BBOB-benchmarking the DIRECT global optimization algorithm. In Rothlauf [2009], pages 2315–2320. ISBN 978-1-60558-505-5.

Petr Posik. BBOB-benchmarking the generalized generation gap model with parent centric crossover. In Rothlauf [2009], pages 2321–2328. ISBN 978-1-60558-505-5.

Petr Posik. BBOB-benchmarking two variants of the line-search algorithm. In Rothlauf [2009], pages 2329–2336. ISBN 978-1-60558-505-5.

Petr Posik. BBOB-benchmarking the Rosenbrock's local search algorithm. In Rothlauf [2009], pages 2337–2342. ISBN 978-1-60558-505-5.

M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. *Large Scale Nonlinear Optimization*, pages 255–297, 2006.

Kenneth Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, 1997.

Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., 2005. ISBN 3540209506. URL http://portal.acm.org/citation.cfm?id=1121631.

K.V. Price. Differential evolution: a fast and simple numerical optimizer. In *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*, pages 524–527, 1996. doi: {10.1109/NAFIPS.1996.534790}.

I. Rechenberg. *Evolutionsstrategie*. Friedrich Frommann Verlag (Günther Holzboog KG), Stuttgart, 1973a.

Ingo Rechenberg. *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973b.

Raymond Ros. Benchmarking the BFGS algorithm on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2409–2414. ISBN 978-1-60558-505-5.

Raymond Ros. Benchmarking the BFGS algorithm on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2415–2420. ISBN 978-1-60558-505-5.

Raymond Ros. Benchmarking the NEWUOA on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2421–2428. ISBN 978-1-60558-505-5.

Raymond Ros. Benchmarking the NEWUOA on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2429–2434. ISBN 978-1-60558-505-5.

Raymond Ros. Benchmarking sep-CMA-ES on the BBOB-2009 function testbed. In Rothlauf [2009], pages 2435–2440. ISBN 978-1-60558-505-5.

Raymond Ros. Benchmarking sep-CMA-ES on the BBOB-2009 noisy testbed. In Rothlauf [2009], pages 2441–2446. ISBN 978-1-60558-505-5.

Raymond Ros and Nikolaus Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 296–305. Springer, 2008a. ISBN 978-3-540-87699-1.

Raymond Ros and Nikolaus Hansen. A simple modification in CMA-ES achieving linear time and space complexity. Research Report 6498, INRIA, jun 2008b. URL http://hal.inria.fr/inria-00270901/en.

Franz Rothlauf, editor. *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*, 2009. ACM. ISBN 978-1-60558-505-5.

M. Schumer and K. Steiglitz. Adaptive step size random search. *Automatic Control, IEEE Transactions on*, 13:270–276, 1968.

Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

Michèle Sebag and Antoine Ducoulombier. Extending population-based incremental learning to continuous search spaces. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 418–427, London, UK, 1998. Springer-Verlag. ISBN 3-540-65078-4.

Yun-Wei Shang and Yu-Huang Qiu. A note on the extended rosenbrock function. *Evol. Comput.*, 14(1):119–126, 2006. ISSN 1063-6560. doi: http://dx.doi.org/10.1162/106365606776022733.

D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656, 1970.

Y. Shi and R. Eberhart. Modified particle swarm optimizer. In *The 1998 IEEE International Conference on Evolutionary Computation, ICEC'98*, pages 69–73, 1998.

Y. Shi, RC Eberhart, E.D.S.I.T. Center, and IN Carmel. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, 1999.

Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173, March 2008. doi: 10.1016/j.ejor.2006.06.046.

James Spall, Stacy Hill, and David Stark. Theoretical framework for comparing several stochastic optimization approaches. In *Probabilistic and Randomized Methods for Design under Uncertainty*, pages 99–117. Springer, 2006. URL http://dx.doi.org/10.1007/1-84628-095-8_3.

James C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 2003. ISBN 0471330523.

R. Storn. On the usage of differential evolution for function optimization. In *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*, pages 519–523, 1996. doi: {10.1109/NAFIPS.1996.534789}.

R. Storn and K. V. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, Berkeley, CA, 1995. URL ftp://ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997. ISSN 0925-5001. doi: {10.1023/A:1008202821328}. URL http://dx.doi.org/10.1023/A:1008202821328.

P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, Singapore and KanGAL, IIT Kanpur, India, Singapore, May 2005.

Thorsten Suttorp, Nikolaus Hansen, and Christian Igel. Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 75(2):167–197, 2009.

Aimo Törn and Antanas Žilinskas. *Global optimization*. Springer-Verlag New York, Inc., New York, NY, USA, 1989. ISBN 0-387-50871-6.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006. ISSN 0025-5610. doi: http://dx.doi.org/10.1007/s10107-004-0559-y.

P. Patrick Wang and Der-San Chen. Continuous optimization by a variant of simulated annealing. *Comput. Optim. Appl.*, 6(1):59–71, 1996. ISSN 0926-6003.

Anatoly Zhigljavsky and Antanas Žilinskas. *Stochastic Global Optimization*, volume 9 of *Optimization and Its Applications*. Springer, 2008. ISBN 978-0-387-74022-5.